GET MORE, FOR LESS!

Interested in getting a digital subscription to another IEEE Computer Society magazine? With a digital subscription, you’ll get videos, podcasts, and interactive links to the latest articles—all delivered via email at the half-year member subscription price of just $19.50 per magazine. This rate expires 10 August 2015. You’ll also get free access in the Computer Society Digital Library to previous issues of the magazines you subscribe to.

Subscribe now by clicking “Subscription options” under the magazine titles at computer.org/subscribe.
15  Customizing Unique Solutions for Software

38  When the Physical and the Cyber Become One

44  3D Models And Processing Tools: Bridging the Gap
Digitally-Enhanced Cosmetics? It’s Possible!

6 Spotlight on Transactions: Do Computers Have Personality? BJÖRN SCHULLER

8 Editor’s Note: DevOps

9 DevOps: Breaking Barriers to Benefit Bottom Lines MARGO MCCALL

12 The Five Properties of Successful Architectural Oversight NICK ROZANSKI

15 Bespoke Infrastructures DIomidis Spinellis

18 Cloud-Based Software Crowdsourcing WEI-TEK TSAI, WENJUN WU, AND MICHAEL N. HUHNS

24 Injecting Value-Thinking into Prioritization Decisions JANE CLELAND-HUANG

29 Is Amazon Becoming the New Cool Software Company for Developers? DAVID BERNSTEIN

32 Toward a History of Social Computing: Children, Classrooms, Campuses, and Communities JOY RANKIN

35 Silver Bullet Talks with Nate Fick GARY MCGRAW

38 The Human Intranet—Where Swarms and Humans Meet JAN M. RABAЕY

44 Demystifying Quadrilateral Remeshing DANIELE PANOZZO

52 The Future of Work RICHARD MATEOSIAN

55 Beauty Technology: Body Surface Computing KATIA VEGA AND HUGO FUKS

72 Life and Tech: Kideo Games BRIAN KIRK

Departments

4 Magazine Roundup
60 Career Focus: Software Engineering Economics
61 Career Opportunities
The IEEE Computer Society’s lineup of 13 digital magazines covers cutting-edge computing topics ranging from software design and computer graphics to Internet computing and security and privacy, from scientific applications and machine intelligence to cloud migration and microchip manufacturing. Here are some highlights from recent issues.

**Computer**

Current conditions create multiple opportunities for hackers and other malicious actors to commit cybercrimes ranging from fraud to identity theft to denial-of-service attacks. For the March 2015 issue of *Computer*, cybersecurity experts present emerging technologies and methodologies to mitigate a broad array of cyberthreats.

**IEEE Software**

Using the **continuous delivery** approach, engineering teams produce valuable software in short cycles to ensure it can be reliably released at any time. *IEEE Software’s* March/April 2015 issue describes how Irish gambling bookmaker Paddy Power adopted continuous delivery and the enhanced capabilities that resulted.

**IEEE Internet Computing**

Network developers, the control community, and middleware researchers are providing the building blocks toward a global Internet of Things. How to combine such building blocks in the design, implementation, and validation of IoT software is the focus of *IEEE Internet Computing’s* March/April 2015 special issue.

**Computing in Science & Engineering**

Improved simulation methods and hardware advances are changing common biomolecular simulation methodologies. In *CiSE’s* March/April 2015 issue, a team from the University of Utah observes that certain aspects of biomolecular simulation workflows are facilitated by access to more heterogeneous computational resources.
For 35 years, the IEEE Symposium on Security and Privacy has provided a venue for experts to share ideas. The March/April 2015 issue of IEEE S&P brings together presentations at the most recent symposium on topics including system security vulnerabilities and challenges in large-scale untrusted systems.

**IEEE Cloud Computing**

Powered by large datacenters with numerous virtualized servers, high-bandwidth networks, and cooling and power systems, cloud computing already constitutes more than one percent of the world’s electricity use. IEEE Cloud Computing’s January/February 2015 issue looks at current approaches for cloud processing energy efficiency.

**IEEE Computer Graphics and Applications**

With the decline of a one-size-fits-every-customer strategy, manufacturers are recognizing the need to reinvent how their factories operate, sparking the Industrie 4.0 and Industrial Internet initiatives based on cyber-physical systems and the IoT. IEEE CG&A’s March/April 2015 issue focuses on visual challenges for these initiatives.

**IEEE Intelligent Systems**

The field of predictive analytics uses statistical or machine-learning methods to make predictions about future or unknown outcomes. With big data, predictive analytics must run faster and more accurately using larger, more heterogeneous sources. The March/April 2015 issue of IEEE Intelligent Systems surveys this expanding field.

**IEEE MultiMedia**

Sonification is especially suited for studying complex, dynamic datasets, especially the large and multivariate time-based datasets that appear in climate science. IEEE MultiMedia’s January–March 2015 issue offers sonification designs that explore conceptual links between climate science and sound.

**IEEE Annals of the History of Computing**

Begun in 1969, a 20-year series of ARPANET maps produced by Bolt Beranek and Newman signifies the earliest efforts to represent a central piece of the modern Internet. IEEE Annals’ January–March 2015 issue looks at this project, which—though initially met with skepticism—proved a major success in computer networking.

**IEEE Pervasive Computing**

To achieve their full potential, pervasive computing technologies must allow authorized users easy access to data and services while making access extremely difficult for unauthorized users with bad intentions. The January–March 2015 issue of IEEE Pervasive considers privacy and security from the perspective of pervasive computing.

**IT Professional**

An ISO/IEEE 11073 personal health device system enables legacy healthcare devices to transmit vital sign data to network application-hosting device. Authors in the January/February 2015 issue of IT Pro argue that standardizing existing healthcare devices for interoperability will require redesigning current hardware and software.

**IEEE Micro**

Mobile CPUs are impacting Web browsing performance and energy consumption. An article in the January/February 2015 issue of IEEE Micro argues that achieving energy-efficient mobile Web browsing requires considering both CPU and network capabilities, looking beyond individual components and rather at the full system.

**Computing Now**

The Computing Now website (http://computingnow.computer.org) features up-to-the-minute computing news and blogs, along with articles ranging from peer-reviewed research to opinion by industry leaders.  

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.
We all have specific personality traits that are reflected in our behavior (or at least those we pretend or appear to have), and we often have strong opinions about the personalities of those surrounding us. But what about computers? Do computers have a personality, or can they at least model different types of personalities? Can they perceive or understand the personalities of their human users? Do Computers Have Personality?

Alessandro Vinciarelli and Gelareh Mohammadi pose this question in “A Survey of Personality Computing” (IEEE Transactions on Affective Computing, vol. 5, no. 3, 2014, pp. 273–291), which explains that the computing industry’s interest in personality has risen steadily since the mid-2000s due to the increasing amount of personal information available through social networks, the possibility of collecting behavioral evidence through smartphones, and the idea of providing machines with social and affective intelligence so they can interact with humans. As such, “personality is relevant to any computing area involving understanding, prediction, or synthesis of human behavior.”

The paper introduces personality computing, explaining that “several works [in the literature] investigate the interplay between personality and computing by measuring the link between [personality] traits and use of technology.” Thus, users externalize their personality through the way they use technology, and personality traits can be indicative of users’ behavior. For example, certain traits can predict whether a user will activate a blog or will use a mobile phone in public spaces.

According to the authors, computing domains concerned with personality consider three main problems: automatic personality recognition (APR; inferring the actual personality based on the individual’s observed behavior), automatic personality perception (APP; predicting a personality attributed to an individual by others), and automatic personality synthesis (APS; generating artificial personalities through embodied virtual agents). The distinction between APR and APP is worth noting in terms of terminology, as affective computing often grapples with differentiating between what’s true and what’s perceived, especially where emotion is concerned.

APR and APP use personality analysis methods rooted in pattern recognition. Depending on the analysis modality—such as video, speech, or social media behavior data—features like acoustic vocal cues or facial expression parameters are extracted. Then, statistical machine-learning methods such as the support vector machine model can be used to train a recognition or prediction model. New, previously unseen behavior can then be analyzed for personality cues based on the extracted features.

Two approaches prevail in APS. In machine learning, labeled data is collected from human behavior to train a machine to produce behavior fitting the current situation that induces the target personality in the human observer. In rule-based systems, specific parameters such as pitch, intensity, speaking rate, length and frequency of pauses, hesitations, and word choice are fitted by rules to the modeled personality. For virtual agents, parameters also
include body movements, gestures, and facial behavior including eye contact and expression.

The authors show that reasonable APR results can be obtained by analyzing text and nonverbal and spoken communication, as well as through computer gaming, social media, and wearable device data, and by considering the user's speaking activity, proximity to others, and movements. Similar findings were reported for APP through nonverbal behavior and social media exploration. With APS, certain personality traits can be successfully simulated through speech or facial expression synthesis. Accordingly, virtual agents and robots might already have a perceivable personality. So it seems that computers can understand and show personality, without actually having one.

With the increasing flow of data from social media and elsewhere, personality computing will help mine the digital traces people leave online, make sense of social media users, target advertisement campaigns to the right potential customers, or tune retrieval technologies to users' personality. Personality computing is also likely to play an important role in technologies aimed at detecting personality disorders. Finally, human–computer interaction (HCI) can adopt personality computing technology to better predict users' needs and preferences, helping machines interact with humans more realistically.

BJÖRN SCHULLER is a professor of complex and intelligent systems at the University of Passau, Germany and a senior lecturer in machine learning at Imperial College London, UK. Contact him at schuller@ieee.org.
EDITOR'S NOTE

DevOps

When solicited to write something on this issue’s theme, a colleague responded, “DevOps. Ughh, that word just makes me cringe (it’s like one of those unword words, you know?).”

And the term does crystalize an annoying tendency in tech and biz speak to create acronyms (like “MOOC” and “FIFO/LIFO”), initialisms (like “IoT” and “SLRs”), and combinations of the two (like “NoSQL”)—shorthand terms ostensibly meant to speed and thus facilitate communication, but just as likely to separate “insiders” from “outsiders” in terms of understanding.

Which is ironic, actually, because the whole point of DevOps (a “development/operations” portmanteau) is to encourage greater cooperation, collaboration, and coordination among people who don’t always speak the same “language,” who may be so narrowly focused on their own role in the software engineering process that they’re blinded to the project’s larger, more important goals.

Not surprisingly, this month’s theme articles draw heavily on IEEE Software: a Pragmatic Architect column focusing on the leadership needed to “ensure that development teams design and build systems in the right way for their stakeholders”; a Tools of the Trade column about bespoke infrastructures to support an organization’s overall goals; and a Requirements column that looks at how developers can prioritize decision making to add product value.

In addition, two articles—one from IEEE Internet Computing and another from IEEE Cloud Computing—consider how the cloud is shaping software development in terms of the opportunities it provides for crowdsourcing (bringing as many stakeholders as is feasible into the process). And from IEEE Annals of the History of Computing comes a short look at social computing, a concept fundamental to any discussion of DevOps.

Beyond this month’s theme, you’ll find an interview with an Afghan War veteran turned cybersecurity expert, from IEEE Security & Privacy; a vision of ubiquitously connected devices, from IEEE Pervasive; a stunningly vivid tour of 3D-modeling techniques, from the March/April 2015 issue of IEEE Computer Graphics and Applications; a review of a book on the future of work, from IEEE Micro; and from our flagship magazine Computer, a whimsical foray into the world cosmetic computing.

Bringing together Computer Society offerings across fields that don’t always speak the same “language” is a primary goal of ComputingEdge. We hope this issue will provide readers a chance to devise their own opportunities for new understanding.

B
DevOps—a merging of “development” and “operations”—is rapidly changing how software is produced and released. By as early as next year, market-research firm Gartner expects DevOps to move from a niche strategy to the mainstream, embraced by fully a quarter of Global 2000 companies.

But why this sudden shift in institutional focus?

What Is DevOps?

It’s been nearly a decade since software developer Andrew Shafer and systems administrator Patrick Debois met at the Agile 2008 Conference in Toronto and formed the Agile Systems Administration Group as a way to resolve conflicts between developers and systems administrators.

Since then, much discussion’s revolved about what DevOps is—and how it’s changing the software production landscape for
enterprises and their developers and operations employees.

Sanjeev Sharma and Bernie Coyle’s *DevOps for Dummies*, an IBM Limited Edition publication, describes DevOps as “an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to more quickly seize market opportunities and reduce the time to include customer feedback.”

According to Laurie Wurster, research director at Gartner, “In response to the rapid change in business today, DevOps can help organizations that are pushing to implement a bimodal strategy to support their digitalization efforts. Digital business is essentially software, which means that organizations that expect to thrive in a digital environment must have an improved competence in software delivery.”

**Why DevOps?**

DevOps adoption is being driven by growing use of agile development processes and methodologies, demand for faster production releases, virtualized and cloud infrastructure capabilities, and increased exploitation of datacenter automation and configuration management tools.

Concurrent with slowly changing legacy systems that function as systems of record, consumers—and investors—now expect enterprises to rapidly deliver easy-to-use forward-facing systems that facilitate user engagement.

Commonly, enterprises offer only one or two systems of record releases yearly, but systems of engagement must be continually updated to address customers’ changing needs. And because the two are linked, systems of record must also be updated with each new engagement systems.
release. This creates pressures that DevOps can help alleviate.

All about People

At its root, say Sharma and Coyle, DevOps is a cultural movement that’s all about people. DevOps spans the full range of organizational stakeholders: business owners; architecture, design, and development engineers; those in charge of quality assurance, operations, and security; even enterprise partners and suppliers. Excluding any stakeholder can lead to incomplete DevOps implementation.

To summarize Sharma and Coyle, even the most efficient processes or automated tools are useless unless they address the full organizational needs of the people who must execute those processes and use those tools.

Thus, building a DevOps culture is critical. Such a culture is characterized by close collaboration across roles, a focus on business rather than departmental objectives, trust, and valuing learning through experimentation. But achieving this requires bringing together people from different backgrounds, with different experiences and predispositions—a major challenge for most organizations.

Team Building Is Crucial

Organization leaders must work with teams to build this culture, which rewards collaboration, facilitates sharing, and removes barriers to cooperation.

Typically, for example, operations teams are rewarded for uptime and stability, while developers are measured on new features delivered. These different interests can pit the groups against one another. But if developers and operations teams share responsibility for delivering new features quickly and safely, these differences can be reduced.

The DevOps trend goes beyond implementation and technology management to a deep focus on how to effect positive organizational change. “With respect to culture, DevOps seeks to change the dynamics in which operations and development teams interact,” notes Wurster. “Key to this change are the issues of trust, honesty, and responsibility. In essence, the goal is to enable each organization to see the perspective of the other and to modify behavior accordingly, while motivating autonomy.”

The result: more efficient procedures, products that please stakeholders and customers—and, along with these, strengthened profits across the board.

To find out more, attend the complimentary 21 April DevOps Unleashed event in Mountain View, California. Visit http://www.computer.org/devops for details.

Margo McCall is editorial content manager for the Computer Society’s Computing Now destination site (http://computingnow.computer.org). Have an idea for an article or wish to contribute? Contact Margo at mmccall@computer.org.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.
The Five Properties of Successful Architectural Oversight

Nick Rozanski

ARCHITECTURAL LEADERSHIP is probably the most important part of a software architect’s role. A key aspect of architectural leadership is architectural oversight, which helps ensure that development teams design and build their systems in the right way for their stakeholders—the people who have an interest in the system’s success. Unfortunately, “the right way” means different things to different people.

Obviously, a system’s architectural components must meet its functional requirements. For example, a sales-ordering system will probably need order entry screens, an order manager, and an order database. These components will need to be wired correctly so that the system can perform correctly. New orders will need to flow from the order entry screens to the order database via the order manager.

However, the system will also need to exhibit the right quality properties, such as scalability, resilience, and security (often called nonfunctional requirements). Orders should be accepted only from authenticated customers, and the system should be able to scale to adapt to the volume at busy times. It’s necessary—and often more difficult—to put as much thought into these architecture aspects as you would into functional and information architecture.

Architectural oversight is about keeping a watchful eye on projects as they move from concept and architecture to building, testing, and deployment, ensuring adherence to the original vision and architecture. (And if not, ensuring there are good reasons for this and that architectural changes are properly evaluated, communicated, and agreed upon.) Overseeing in a way that ensures that the system has the right architecture can easily become overly bureaucratic or poorly leveled (for example, focusing on some characteristics in detail while ignoring others). Worst of all, this process can fail to ensure that the delivered systems are sound and fit for the purpose.

Effective architectural oversight has five important properties: it must be timely, objective, systematic, constructive, and, most important, pragmatic (see Figure 1). Here, I examine each property to understand what it means, why it’s important, how you can achieve it, and what might happen if you don’t.

For this article’s purposes, I’ll assume you’re an architect with several systems in your portfolio, including new builds or systems undergoing substantial changes. You might be the architect of a collection of systems that perform a particular business function, or of the systems in one location. However, if your role is different—for example, you’re a
solution architect looking after a single system or an enterprise architect responsible for the whole systems landscape—much of what I describe here will still apply to you.

**Timely**
Architectural oversight must occur at the right times. Get involved in development projects early, when the architecture is still being formulated and important decisions are being made. Such decisions include

- what the system’s fundamental structure will be,
- how the system will manage data and use other systems’ data,
- how it will scale and be resilient and secure, and
- how it will use advanced or unfamiliar technologies or approaches.

These decisions are difficult to overturn or change later during the project. For example, if your team starts out building a monolithic system, turning it into a collection of loosely coupled, semi-independent components will be a lot of work.

If you’re using waterfall or iterative development, expect to scrutinize the architecture at the start of high-level design. There are strong arguments for you to be involved even earlier, while the architecture is still being formulated. This lets you give your team the benefit of your experience and knowledge of what works (and what doesn’t).

If you’re using agile methods, your interaction model will be a little more fluid. Most agile methods include some architecture envisioning early on, although this might be less precise than it would be for waterfall or iterative development. The architecture will likely evolve and crystalize during subsequent agile sprints or iterations. You’ll need ongoing oversight to ensure that the architecture is moving in a sensible direction and that important aspects aren’t being ignored.

Architectural oversight performed at the wrong time (for instance, after the big architectural decisions have already been made and acted on) has little chance of improving the system’s architecture.

**Objective**
Architectural oversight must be based on clearly stated principles, guidelines, and patterns, rather than on a subjective opinion. We all have our own ideas about how best to build software systems. An old joke says that if you put two architects in a room, they’ll come up with at least three ways to design a system. Although there are common architectural principles and patterns that we all work toward, there’s more than one “good” way to build a system to meet a given set of functional requirements and quality properties. In these cases, the choice between architectures will come down to the architect’s experience, preferences, and (sometimes) prejudices, which might differ from yours.

The goal of objectivity is to come to the same overall conclusions about the architecture regardless of who does the actual oversight and, specifically, how that person would have designed the architecture. This helps ensure a fair, objective process. A good way to achieve objectivity is to have a set of written guidelines, standards, and patterns that project teams can follow (or are required to follow). These guidelines should quickly lead team members to the right decisions, while leaving room for innovation and creativity.

**Systematic**
Architectural oversight must follow well-defined, repeatable processes whose objectives and outcomes are clearly understood. Systematic oversight, whether lightweight or thorough, will follow a defined sequence of steps with known inputs and outputs to achieve a well-defined goal in a finite (and hopefully relatively brief) time period. The participants are known, and everyone understands their role and responsibilities. The tasks are clearly defined, and all participants clearly understand the overall objective.

Ensure your outputs are recorded in enough detail so that they are understood, can be acted on, and are clearly communicated to your stakeholders. Itemize actions and close them when complete, and record decisions along with their rationale (why the decision was made) and implications (what must happen next). Clearly document exceptions involving noncompliance with policy, standards, or general good practice, along with the reasons for the exceptions.

If your oversight isn’t systematic,
your stakeholders won’t clearly understand who’s involved, why the oversight is being done, when it was started or finished, and its outcomes.

Constructive
Architectural oversight must lead to better architecture. Constructive oversight results in real, significant, and valuable change. In practice, this often means that the resulting architecture is more scalable, resilient, highly available, and secure. These qualities often receive less attention than they deserve when developers are ensuring that a system meets its functional requirements. However, a system that is unreliable, is too slow, or has security holes won’t be viewed as a success and might even be abandoned.

The most constructive oversight leads to specific architectural changes or improvements. For instance,

- functional components might be added,
- the system might be better designed for resilience or high availability,
- the system might be given scale-out or scale-up features so that it can continue to perform under high loads, or
- security features might be improved to better control access to sensitive functions or data.

If architectural oversight doesn’t lead to specific improvements, your stakeholders will view it as a waste of time or a bureaucratic form-filling exercise they must endure, rather than an important part of software development.

Pragmatic
Architectural oversight must take into account real-world constraints such as time, cost, and the availability of skills, without diluting the oversight’s purpose or effectiveness. You’ll most often have to deal with time and cost constraints. There’s no point in insisting on an architectural capability or feature if implementing it would substantially overrun or blow the project’s budget. You must be ready to compromise while ensuring the architecture’s overall integrity.

Start by ensuring that you and your stakeholders have a common understanding of the business importance of the system you’re looking at. This will help you pitch your oversight at the right level. You can then assess the benefits and risks to help you come to a decision.

For example, a proposed business-critical system with no disaster recovery capabilities would be a high risk because a significant incident could shut down the business. This aspect should be remediated before the system goes into production. On the other hand, it’s probably reasonable to not implement such a feature for a proof-of-concept system that will be rewritten anyway if it’s successful.

In any case, don’t expect everyone to understand and agree on all the details of a system’s architecture, especially at the early stages of a project. If some architecture aspect is still being developed, focus on tracking this to a decision rather than labeling the system as “non-compliant” in some way.

Don’t aim for architectural perfection or demand unnecessary architectural capabilities. Be prepared to compromise when it’s right to do so, but ensure that these compromises are understood, agreed on, documented, and communicated to all stakeholders.

NICK ROZANSKI is a lead architect in a major UK bank’s Chief Technology Office. Contact him at nick@rozanski.org.uk.
Bespoke Infrastructures

Diomidis Spinellis

IN THE 1920s, the Ford Motor Company embarked on an ill-fated attempt to establish an industrial town in an Amazon rainforest as a way to secure a cultivated rubber supply for its cars’ wheels. At the time, it already owned ore mines, forests, and a steel foundry to produce the raw materials for its cars; today, it buys from external suppliers, even for its cars’ electronic control units. How do these two phases of the automotive industry’s history relate to the way we currently develop and adopt infrastructure in our profession?

Infrastructure developed within your organization for its own internal use can take many forms: operating systems; compilers; programming languages; version control systems; platforms for building, testing, and continuous integration; database management systems; application development frameworks, game engines; or utility libraries. Bespoke infrastructures can also extend to methods for doing work, such as the development process, code reviews, workflows, code style rules, and testing and integration practices.

The Case For
The obvious reason for creating a bespoke solution is that it can be tailored to fit your organization’s unique needs. For example, you can optimize the design of a bespoke database management system or cache server to fit exactly your organization’s load and query profile. Aggressively tailored solutions can run circles around offerings that try to please everyone, supporting features particular to your organization’s unique needs: a programming language construct, a database column type, or a game engine interaction style.

Then there’s the flexibility: as the owner of the infrastructure, you decide where it’s going. If you want to add a new feature or fix a bug, you
devote the required resources, and, presto! Your wish is fulfilled. In contrast, if you adopt a commercial offering, you can only hope that the vendor moves in the direction you want; if you work with an open source solution, you have to coordinate with its developers (and sometimes jump through multiple hoops) to integrate your changes upstream.

Put simply, bespoke infrastructures allow your organization to innovate and keep the fruits of any findings to itself, which can provide it with tactical or even strategic advantages over the competition. As examples, consider the bespoke database and caching solutions that allow big social networking companies to drink data from a fire hose and the awesome proprietary data-center infrastructures developed by the largest cloud service providers. Even if the benefits of a bespoke infrastructure are dubious, its mere existence can serve as a selling point or a differentiator in the market.

**The Case Against**

Proprietary infrastructure is only known within the organization that hosts it. Consequently, new employees face a significant hurdle before they can become productive and stop inundating their colleagues with questions. Contrast this with the case of a widely used offering that lets newcomers add value to the organization from day one by folding in their relevant knowledge, experience, and improved practices. The use of a bespoke infrastructure imposes its own vocabulary, hindering the informal communication of developers with colleagues in other organizations. Along the same lines, users of a bespoke solution won’t be able to reach out to the global online community for answers and support—a convenience that we take for granted today.

Maintenance is another issue. Let’s assume that, at the time you set up your bespoke infrastructure, it suited your organization better than any alternative. However, to paraphrase Robert Anton Wilson, it takes just two years for some brilliant software to turn into a nightmare without changing a single line of code. Unless aggressively maintained and developed, bespoke infrastructures can easily fall behind the state of the art. What was once a nimble trailblazer opening new directions for your organization can quickly become a dinosaur that holds progress back. I’ve heard developers complaining that their organization’s bespoke development tools, probably once a source of pride, are in such a state of disrepair that they spend more time waiting for their environment to work than the time they invest in actually writing code.

Then come the development and support costs, which will include not only the (typically highly paid) engineering time needed to bring the infrastructure to life, but, just as importantly, management distraction during both its early days and its (inevitably capricious) ending ones. Add to this the opportunity cost of depriving other profitable projects of engineering resources, and the price can really go up.

But the problems don’t stop here. Given that infrastructure is critical to operations, the owners of bespoke solutions can (often unintentionally) hold the organization ransom to secure cozy working arrangements.

**It takes just two years for some brilliant software to turn into a nightmare without changing a single line of code.**

This drives down morale and encourages empire building by piling new layers of bespoke stuff on top of existing ones. As you might expect, such vested interests in an organization stand in the way of looking at better alternatives, and the organization misses out on the benefits of the latest and greatest technology.

Finally, consider developer mobility. On one hand, developers who for years have been writing code in your organization’s obscure programming language that no one else uses will find it difficult to get an offer that will lure them away. On the other, the smart people who work with your niche infrastructure will quickly realize that it negatively affects their career prospects and will start looking for alternatives. Thus you’ll end up working only with those unfortunate souls who have nowhere better to go.

**A Balancing Act**

Maintain a healthy amount of skepticism regarding homebrew solutions: the cards are stacked against the adoption of infrastructure that’s “not invented here.” By definition, bringing in such infrastructure
means change, and this triggers people’s conservative instincts. Developers who have learned to use the bespoke tool or library will have to learn the new one, and, worse, those who developed it will have to find other ways to contribute.

It’s impossible to break new ground with established solutions, so the need to come up with a never-ending stream of bespoke solutions might just be the cost of doing business at the frontier. Yet, the problem may not be in creating and using these infrastructures, but in not letting them go when they’ve served their purpose.

You might hear arguments about the investment put into a bespoke infrastructure’s development. Given that this is a sunk cost, it shouldn’t influence your decision either way. Rather, you should simply consider the relative merits of the two solutions, the cost of the alternatives, and any switching costs. Sadly, misplaced loss aversion regarding a sunk cost often taints an organization’s judgment.

If universally available tools don’t quite fit the bill, consider customizing a general-purpose solution to your needs. Thankfully, modern technologies are often easily customizable via myriad configuration options, plugins, and modules. (Often to the point of absurdity; consider the 12,000 theme downloads available on eclipsecolorthemes.org.) Look for existing customizations before launching your own.

Another approach is to adopt an open source tool and improve it to address your organization’s requirements. Then, cooperate with the tool’s developers to contribute your changes back to the community. This isn’t just out of altruism; feeding your changes back upstream ensures that they remain part of the tool in the future.

Finally, when called to make a choice, consider that the trend is toward a transition from bespoke infrastructures to widely used, general-purpose technologies. I’ve seen this transition happening in many organizations, often with pain and regret for the earlier decision to follow the bespoke solution sirens.

When you design infrastructures, train your instinct to go with the flow: adopt and build on the best and greatest technologies used by your community.

DIOMIDIS SPINELLIS works at Google as a site reliability engineering software engineer. Contact him at dspin@google.com.

Cloud-Based Software Crowdsourcing

Wei-Tek Tsai • Arizona State University
Wenjun Wu • Beihang University
Michael N. Huhns • University of South Carolina

In addition to providing large-scale, highly available computational resources, clouds also enable a new methodology for software development via crowdsourcing, in which crowd participants either collaborate or compete to contribute software. Using a crowd to develop software is predicted to take its place alongside established methodologies, such as agile, scrum, pair programming, service-oriented computing, and the traditional waterfall.

Crowdsourcing software development, or software crowdsourcing,1 is an emerging software engineering approach. Software development has been outsourced for a long time, but using a cloud to outsource it to a crowd of developers is new. We’ve found that all software development tasks can be crowdsourced, including requirements, design, coding, testing, evolution, and documentation. Software crowdsourcing practices blur the distinction between users and developers, and follow the cocreation principle — that is, a regular user becomes a codesigner, codeveloper, and comaintainer. This is a paradigm shift from conventional industrial software development, with developers distinct from users, to a crowdsourcing-based, peer-production software development, in which many users can participate.

A cloud provides a scalable platform with sufficient resources, including computing power and software databases, for a large crowd of developers. With emerging cloud software tools such as DevOps (a portmanteau of development and operations) and large-scale software mining, a cloud significantly reduces the amount of manual labor needed to set up software production environments and empowers peer developers to perform software crowdsourcing tasks efficiently in design, coding, and testing.

Based on its organizational style, software crowdsourcing can be either competitive or collaborative. In competitive crowdsourcing, only winning participants are rewarded. TopCoder (www.topcoder.com), an online programming contest site, is an example of this approach. In collaborative crowdsourcing, people cooperate with each other on various aspects, including funding, concept development, user interface design, code, test, and evaluation. AppStori (www.appstori.com) represents this approach. The process design (such as activities, duration, and number of participants), support infrastructure, and software projects are different for these two approaches.

Crowdsourcing Development Processes
Crowdsourcing can be incorporated into conventional software development processes such as waterfall or agile and can contribute to any software development phase.2 It can be used with most design techniques, such as object-oriented, service-oriented, and user-centered design (UCD); software-as-a-service (SaaS); and formal methods. For example, TopCoder directly supports the following process: conceptualization, specification, architecture, component production, application assembly, certification, and deployment. Each can be crowdsourced competitively.
Crowdsourcing Goals
Crowdsourcing’s benefits accrue from organizations attempting to achieve the following goals:

- **Quality software.** Such software comes from competent participants who try to outdo their peers in submitting innovative concepts, design, code, or tests.
- **Rapid acquisition.** Crowdsourcing organizers can post a competition hoping to find that something similar has already been developed.
- **Talent identification.** An organizer might be interested in identifying talented developers, as demonstrated by their performance in competitive efforts.
- **Cost reduction.** A crowdsourcing organizer can acquire software at a low cost owing to the need to pay only winners, and could even pay below-market costs, given that participants might seek reputation rewards rather than monetary ones.

Other goals include solution diversity, idea creation, broadening participation, marketing, and participant education, such as encouraging people to use or learn specific tools. To ensure a good outcome from software crowdsourcing, organizations can leverage a cloud infrastructure to accelerate the process of setting up the development environment and enabling distributed and large-scale development by a highly dynamic community.

The turnout for a software crowdsourcing event varies due to crowdsourcing’s open and online nature. A project with a high reward will attract a relatively large number of crowd workers. Moreover, computational overhead and data traffic for different software development tasks lead to spikes in the resources that crowdsourcing activities require. To address this issue, cloud-based software crowdsourcing combines advantages of both paradigms: pay-for-use and pay-for-performance. An elastic cloud computing resource lets crowdsourcing organizers cope with fluctuations in the numbers of participants as well as computational workloads that occur with crowdsourcing competition activity. For example, the largest crowdsourcing site for data analytics, Kaggle (www.kaggle.com), adopted Windows Azure to run software services and host crowdsourcing contests.

**Industrial Software Crowdsourcing**
Several websites have been established to support crowdsourcing, such as TopCoder, uTest (www.utest.com), AppStori, oDesk (www.odesk.com), and mob4hire (www.mob4hire.com). Furthermore, most of the industrial software giants are actively engaged in crowdsourcing, including Microsoft and Oracle. Microsoft used crowdsourcing for Windows 8 development by starting blogs, crowdsourcing mobile devices for Windows 8, and offering US$100,000 for security testing. Oracle also adopted crowdsourcing for its customer relationship management projects.

**Architecture and Models**
The distributed development of a software system by a crowd requires the guidance of a reference architecture and models for the development process.

**Software Crowdsourcing Architecture**
Different software crowdsourcing processes can have different needs, but also share some commonalities. Such common themes for software crowdsourcing processes include

- a cloud service management dashboard for system administrators and software crowdsourcing organizers;
- collaboration and communication tools, such as a distributed blackboard system where each party can participate in discussions;
- participant ranking and recommendation tools;
- software development tools for modeling, simulation, code editing and compilation, design notation and documentation, and testing;
- cloud payment and credit management tools; and
- a repository of software development assets, such as modules, specifications, architectures, and design patterns.

All these elements are encapsulated in the reference architecture for a cloud-based software crowdsourcing system (Figure 1). Metaphorically, we can regard this architecture as synergy between two clouds — machine and human — toward the ultimate goal of developing high-quality and low-cost software products. With the properly specified platform-as-a-service (PaaS) recipes, software project managers can establish a customized cloud software environment to facilitate the following software crowdsourcing process:

1. A manager uses software networking and collaboration tools to design a reward mechanism to motivate crowd workers. Depending on the software product’s value and the development scale, a project manager can specify the appropriate budget to attract as many talented developers as possible and provision computing resources to sustain their activities.
2. The manager uses project management tools to coordinate development tasks among the crowd workforce by ranking individuals’ expertise and matching their skills with the different task levels and types.
3. Toward a specific software project, a manager sets up a virtual system platform with all the necessary software development gears to assist crowd workers with their tasks.
4. Both the platform and its workflow can operate elastically to yield cost-efficient resource utilization.

We discuss the cloud tools in detail in a subsequent section.

Software Crowdsourcing Models
We can characterize software crowdsourcing in terms of the crowd size, software scale and complexity, development processes, and competition or collaboration rules. Formal models for designing and modeling software crowdsourcing can have the following foundations:

- **Game theory.** The nature of contests in competition-based crowdsourcing can be analyzed via game theory. For example, we can determine the reputation reward value based on the number of participants and the reward price; participants are often willing to compete to gain reputation rather than receiving a reward price determined according to a Nash equilibrium.\(^2\) Even the Prisoner’s Dilemma game can be viewed in terms of crowdsourcing (www.dellingadvisory.com/blog/2013/4/10/crowdsourcing-the-prisoners-dilemma).

- **Economic models.** Economic competition models provide strategies and incentives to generate crowd participating and reward-structuring rules for organizers to maximize crowdsourcing returns. The recently developed contest theory\(^7\) introduces new mathematical tools, such as all-pay auction, to describe the synergy among individual efforts, competition prize structure, and product quality.

- **Optimization theory.** Due to the competitive and dynamic nature of software crowdsourcing processes, coordinating unstable virtual teams, optimizing the partition and allocation of development tasks, and balancing costs, time, and quality can be challenging. Thus, the organizer of a crowdsourced software development effort could apply a search-based software engineering approach\(^8\) to address optimization problems.

By using platforms that incorporate open source software repositories, such as Github and commercial crowdsourcing sites, we can conduct experiments to understand software crowdsourcing’s nature and validate theories. Such experiments must model tasks, workers, and costs for effort and resources. We can verify various worker ranking, incentive, and matching algorithms to simulate possible virtual team formation, and to isolate design factors. Both professional engineers and students can be involved in experiments, and investigators can document and make available ontologies of these models, so that others can design new experiments, collect data, and exchange ideas. Several platforms are available, including OW2 Open Source Community and Trustie (http://forge.trustie.net). Simulation techniques, such as NetLogo, that are based on multiagent systems can also play a role in modeling software crowdsourcing behavior.

**Cloud Tool Support**
Given software crowdsourcing’s distributed nature, it needs a powerful...
Cloud-Based Software Crowdsourcing

Cloud tools for supporting software crowdsourcing.

<table>
<thead>
<tr>
<th>Cloud tool type</th>
<th>Name</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software development tools</td>
<td>Chef/Puppet</td>
<td>Streamlines the environment configuration for cloud-oriented software development and testing</td>
</tr>
<tr>
<td></td>
<td>PaperTrail</td>
<td>Supports large-scale system log administration and analysis</td>
</tr>
<tr>
<td></td>
<td>CloudIDE</td>
<td>Web-based IDE that simplifies and integrates cloud software development</td>
</tr>
<tr>
<td>Social networking and collaboration tools</td>
<td>Confluence</td>
<td>A team-based collaboration tool for content creation and sharing</td>
</tr>
<tr>
<td>Project management tools</td>
<td>CloudSpoke on Topcoder</td>
<td>A crowdsourcing website for cloud development that ranks developers’ skills and organizes contests for development tasks</td>
</tr>
<tr>
<td></td>
<td>Github</td>
<td>A project-hosting website that supports collaborative software development, and can serve as code repository for cloud-based software crowdsourcing</td>
</tr>
<tr>
<td></td>
<td>Trustie</td>
<td>An online software development site that integrates project hosting, social networking, and programming education</td>
</tr>
</tbody>
</table>

development environment to facilitate software design, coding, test, and deployment across distributed and heterogeneous infrastructures. All community members should be able to access the same software development environment customized for a specific project. Moreover, numerous submissions from community members must be quickly screened, evaluated, reviewed, and integrated. Thus, software crowdsourcing efforts require enhancements to common software development tools for coding, testing, and deployment to support automatic project building, integration, performance analysis, and security checking. By leveraging cloud computing’s elastic resource provisioning and virtual appliances, crowdsourcing practitioners have started to develop the evaluation pipeline to automate testing and review for quality, security, and coverage, as well as coding standards. A cloud-based integrated development environment (IDE) must have the following features (Table 1 summarizes the various tools).

Software Development Tools
An IDE for crowdsourcing integrates tools for requirements, design, coding, compilers, debuggers, performance analysis, testing, and maintenance. For example, cloud software configuration tools such as Chef (www.opscode.com/chef/) and Puppet (http://puppetlabs.com) let community members establish their own virtualized development environments. MapReduce-based log-management tools, such as PaperTrail (http://papertrailapp.com), support large-scale system log administration and analysis, and help community members resolve software problems and enhance system reliability using log messages.

Social Networks and Collaboration Tools
Facebook, Twitter, wikis, blogs, and similar Web-based tools let participants communicate for sharing and collaboration. For example, organizers can use Facebook profiles to form a virtual crowdsourcing team, even if the participants don’t know each other. A collaborative blackboard-based platform can let participants see a common area and suggest ideas to improve the solutions posted there.

Project Management Tools
Crowdsourcing project management should support project cost estimation, development planning, decision making, bug tracking, and software repository maintenance, all specialized for the context of the dynamic developer community. In addition to these regular functions, it must incorporate ranking, reputation, and award systems for both products and participants. For example, TopCoder introduces a sophisticated ranking scheme, similar to sport tournaments, to rank community members’ skills in software development. Community members often decide to participate in a specific contest if they know the ranking of the participants already enrolled.

Ecosystem for Software Crowdsourcing
Both collaboration- and competition-based software crowdsourcing have ecosystems with significant economic implications. These ecosystems create jobs and establish career paths for developers. They also let organizations start new projects, secure funding, identify talent, and create new products. For example, Apple’s App Store, a website for iOS applications, has an ecosystem with 150,000 contributors and has accumulated more than 700,000 iOS applications in its four years of operation. People contribute their innovations for reputation or money via the store’s micropayment mechanism. With such a large community, several community-based, collaborative platforms have
Crowdsourcing Roadmap

A recent workshop presented the following four-level roadmap for software crowdsourcing.9

**Level 1**
The first level is characterized by individual developers, well-defined modules, small size, limited development time span (less than a few months), quality products, and current development processes such as the ones by AppStori, TopCoder, and uTest. Coders are ranked; websites contain online repository crowdsourcing materials; participants can rank software; and crowdsourcing platforms have communication tools such as wikis, blogs, and comments, as well as software development tools such as an IDE, testing, compilers, simulation, modeling, and program analysis.

**Level 2**
At the second level are teams of people (<10) and well-defined systems; a medium-sized system to be developed, with medium development time (several months to less than one year); and adaptive development processes with intelligent feedback in a common cloud platform where people can freely share thoughts. At this level, a crowdsourcing platform supports an adaptive development process that allows concurrent development processes with feedback from fellow participants; intelligent analysis of coders, software products, and comments; multiphase software testing and evaluation; big data analytics, automated wrapping of software services into SaaS, annotations with terms from an ontology, and cross references to DBpedia and Wikipedia; automated analysis and classification of software services; and ontology annotation and reasoning, such as linking those services with compatible I/O.

**Level 3**
The third level has teams of people (<100 and >10), a large well-defined system, long development time (<2 years), and automated cross-verification and cross-comparison among contributions. A crowdsourcing platform at this level contains automated matching of requirements to existing components, including matching of specifications, services, and tests, and automated regression testing.

**Level 4**
The fourth level consists of a multinational collaboration of large and adaptive systems. A crowdsourcing platform at this level might contain domain-oriented crowdsourcing with ontology, reasoning, and annotation; automated cross-verification and test-generation processes; and automated configuration of the crowdsourcing platform. It might also restructure the platform as SaaS with tenant customization.

**Pilot Software Crowdsourcing Projects**
The University of South Carolina recently started a cloud-based crowdsourcing project to produce the control software for an office robot. The robot behavior will follow a three-step process: perceiving, reasoning, and acting, commonly known as a PRA architecture. Specifically, perceiving occurs using ultrasonic sensors to detect walls and obstacles. Moving to an office is an action that requires reasoning using data collected in the perceiving subsystem. A crowd consisting of students and faculty will develop software for the robot at the granularity of a behavior. The goal is to show that the crowd size need not be large. Developers can act autonomously and collaboratively. To ensure interoperability, software produced will be stateless services, so that they can be independent of each other, and each module will expose a service specification so that specifications can be formally reasoned. Furthermore, software development will be done using a common IDE, a cloud-based repository (such as Bitbucket or GitHub), and a Ubuntu Linux platform; testing can occur either in the cloud or on the actual robot.

been created as incubators, and one such incubator is AppStori, an online crowd-funding community in which people can collaborate to develop iOS apps.

TopCoder, founded in 2001, similarly has an ecosystem with a global workforce of more than 500,000 members. Recently, TopCoder and CloudSpokes (www.cloudspokes.com), which has 72,000 registered members with expertise in cloud software, merged to form the largest crowdsourcing community for cloud-based software development. About 1,100 CloudSpokes projects have been posted online to motivate developers to work on the major cloud platforms, such as Amazon Web Services and Force.com.

Crowdsourcing also needs to focus on workers to keep them happy as the world enters a worker-centered crowdsourcing society. Essential features of a crowdsourcing platform should show past performances and assign proper ratings based on metrics developed by the relevant communities.

Team organization is also important in crowdsourcing, and teams can be self-organized by community or recommendation. Crowdsourcing communities must have objective governing rules to arrange bidding, matching, job security, career paths, project management, and often physical environments for participants. Trust among team members is important because teams might be formed by people who don’t know each other, and it’s necessary to manage trust via multiple strategies such as ranking, recommendation, displayed previous products, and recorded dialogue. Furthermore, a crowdsourcing platform must inform workers whom they work for and what their role in the development effort will be.
Cloud-based software crowdsourcing is a new approach for low-cost, rapid software development, and the existence of large ecosystems has shown that this approach is viable. Moreover, it is a ripe area for research on how a crowd of anonymous developers can produce coherent models, analyses, simulation components, experimental results, and a support environment. Many research directions are possible, including theoretical models, optimization methods, infrastructure support features, and social issues, with a clear roadmap.9

Acknowledgments
The ideas in this article were inspired by participants at the recent Dagstuhl Workshop on Cloud-Based Software Crowdsourcing.9

References

Wei-Tek Tsai is a professor at Arizona State University. His research interests include software as a service, service-oriented computing, and crowdsourcing. Tsai has a PhD in computer science from the University of California, Berkeley. He’s a member of IEEE. Contact him at wtsai@asu.edu.

Wenjun Wu is a professor at Beihang University. His research interests include crowdsourcing, cloud computing, and eScience. Wu has a PhD in computer science from Beihang University. He’s a member of the China Computer Federation (CCF). Contact him at wwj@nlsde.buaa.edu.cn.

Michael N. Huhns holds the NCR Professorship and is chair of the Department of Computer Science and Engineering at the University of South Carolina. He also directs the Center for Information Technology. Huhns has a PhD in electrical engineering from the University of Southern California. He is a senior member of ACM and a fellow of IEEE. Contact him at huhns@sc.edu.
**Injecting Value-Thinking into Prioritization Decisions**

Jane Cleland-Huang

**MOST PROJECTS** have more potential requirements than they have resources to deliver them. Regardless of whether you’re building software to control automobiles, manage patient healthcare records, or calculate driving directions, you need to think long and hard about which features to include in your product and how to prioritize and sequence their delivery. Your decisions will likely have major ramifications on your product’s success and marketability.

People have proposed many approaches for selecting and prioritizing features. The most popular ones are based on various triaging schemes for classifying requirements into high, medium, or low priorities or on assigning points to different features and using the accumulated points to rank the features. However, these techniques are naive and easily influenced by the composition of the stakeholders in the room. Better approaches seek consensus—through bringing together key stakeholders, identifying tradeoffs, articulating value propositions, and reaching agreement. One way to do this is through the approach I describe here, which takes into account the features’ value.

**Story Mapping and Optimizing Value**

Jeff Patton’s latest book describes *story mapping*, an agile practice. Story mapping addresses many problems inherent in a traditional backlog and exposes prioritization decisions to the team as a whole. A visual workspace replaces the backlog’s flat list. Essential stories form a horizontal backbone along a timeline. Stories detailing each step, or simply occurring around the same time, are added vertically beneath the essential stories. Constructing a story map engages project stakeholders in actively planning releases and thrashing out the delivery sequence. Figure 1 shows a simple story map.

Agile release planning is often driven by the goal of identifying and delivering a *minimal viable product* (MVP). Steve Blank and Eric Reis coined this term to describe the deliverable that maximizes feedback from hands-on users at the lowest risk. This makes a lot of sense. Instead of prioritizing user stories by module, you identify a minimal slice of user stories that cut across the system and that, when deployed, bring real value to the customer. In my own agile experiences, we’ve worked exactly this way. Our team has met, thrashed out ideas, and identified an initial product we can place into our users’ hands.

However, release planning has another aspect. Instead of focusing only on validated learning, you should also consider the financial impact of the features you deliver. The book *Software by Numbers*, which Mark Denne and I wrote over a decade ago, introduced *incremental funding* and showed how to sequence *minimum marketable features* (MMFs) so as to optimize project value. MMFs are the smallest unit of func-

It’s a bit naive to believe that a simple rule will always produce the winning solution.
tionality that delivers something of value to the customer. Denne and I measured value in terms of revenue, reduced operating costs, and intangibles such as increased customer loyalty. We showed that smart delivery sequences enabled early revenue that can fund the remainder of the project.

Given multiple objectives, it isn’t always obvious which parts of the system to build first. The goal is to identify a set of features that maximize validated learning potential while returning real value to the customer. To complicate issues, necessary architectural decisions often impact the timeline. For example, you might need to decide whether to build the simplest possible design first and then refactor later to support more functionality, or build the necessary infrastructure early in the project.

For sure, you could follow the agile mantra and always build the simplest solution first. However, it’s a bit naive to believe that a simple rule will always produce the winning solution. The mantra creates a rule of thumb to be followed regardless of whether it’s the best decision for a particular project. This also somewhat contradicts the notion that agile team members should be empowered to make their own informed decisions.

The approach I now describe injects value-thinking into prioritization. These ideas stem from my earlier work on incremental funding; here, I present them in a lightweight format and integrate them into story mapping.

### A Simple Example

Imagine you’re starting Mags-R-Us, a company that acts as the middleman for online magazine delivery. You need to develop a software application to support your business model. You gather together some project stakeholders and brainstorm a set of user stories. These include sign-up, magazine management, dispatch, invoicing, and payments. Your business plan is to build your customer base and iron out problems in your system by launching the first magazine without charging a fee, in order to build a customer base. However, you plan to start charging fees within the first few months. Furthermore, several potential magazines are already lined up to use your delivery service.

You organize your stories into the story map in Figure 1. Yellow indicates user actions; blue indicates system actions. The story map starts when a manager adds a magazine to the system. New editions of the magazine are regularly uploaded. Users can subscribe to the magazine, and the system dispatches it on its planned release dates to all subscribers. The system then invoices the subscribers (perhaps annually), who pay the invoice. In this example, the story map isn’t equivalent to a use case because it captures a more general life-cycle narrative.

---

**FIGURE 1.** A simple story map for Mags-R-Us, a hypothetical business that acts as the middleman for online magazine delivery. Gray indicates the project backbone, yellow indicates user actions, and blue indicates system actions.
The story map serves several purposes. It helps you recognize and write several important user stories, identify the critical ones, and place them in the project backbone (in gray in Figure 1). But you still need to make prioritization decisions. These decisions don’t change the backbone. However, they’ll influence the order in which you build and release the features and their supporting infrastructure.

Consider two delivery sequences; Figure 2 shows the first one, which I call the “big-bang approach.” The first release bundles all the essential functionality to support the Mags-R-Us business. Although this release might prioritize user stories into iterations, the Mags-R-Us system won’t come online until all functionality is delivered.

A simple financial analysis can help determine how choosing this sequence affects the value proposition (see Tables 1 and 2). I make a few simplifying assumptions for the purposes of this illustration. First, most primary user stories (see the first column of Table 1) take one iteration to develop. So, the development period is depicted by the negative dollar amount. Also, revenue can be generated only when invoicing features come online; it will increase once online payments are possible. The cash flow becomes positive only in period 11, taking into consideration the net present value (computed at a discount of 2.5 percent per period).

### TABLE 1

<table>
<thead>
<tr>
<th>Delivery sequence</th>
<th>Development period</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Magazine selection (browser)</td>
<td>–5</td>
</tr>
<tr>
<td>Magazine edition manager (browser)</td>
<td>–5</td>
</tr>
<tr>
<td>Subscription infrastructure</td>
<td>–10</td>
</tr>
<tr>
<td>Customer issue tracker</td>
<td>–5</td>
</tr>
<tr>
<td>Invoicing</td>
<td>–10</td>
</tr>
<tr>
<td>Payments received</td>
<td>–5</td>
</tr>
</tbody>
</table>

* Each figure represents US $1,000.
The second delivery sequence decomposes the MVP into three parts, each delivered in a separate, value-enhancing release (see Figure 3).

Again, a basic financial analysis shows how this sequence plays out (see Tables 3 and 4). Release 1 produces intangible benefits through building the customer base. Revenue is generated at the end of release 2, once invoicing and payments go online. Finally, increased revenue follows release 3 as the full functionality is released. Furthermore, early efforts to grow the customer base generate revenue. This delivery sequence delivers greater value than the first one, despite additional costs for refactoring the dispatch component.

The actual numbers in my example aren’t as important as the process itself. If, for example, building the customer base in advance fails to increase the starting revenue after the full functionality goes online, the second delivery sequence will actually lose money. At the very least, examining release decisions’ financial impact and playing around with various what-if scenarios highlights sensitivity points in the decision making and leads to more informed decisions.

![Figure 3](image-url) This sequence incrementally delivers the MVP in three parts. In this scenario, it delivers greater value than the first sequence (see Figure 2).

**Table 2**

<table>
<thead>
<tr>
<th></th>
<th>Development period</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Cash</td>
<td>-5</td>
</tr>
<tr>
<td>Net cash</td>
<td>-5</td>
</tr>
<tr>
<td>Present value @ 2.50%</td>
<td>-5</td>
</tr>
<tr>
<td>Net present value</td>
<td>-5</td>
</tr>
</tbody>
</table>

*Each figure represents US $1,000.*
In writing this column, I’m reminded of the conversations that took place shortly after the release of Software by Numbers. Some folks claimed that software developers could never make revenue predictions, therefore financially driven approaches would never work. These people really missed the point and obviously saw software development as an isolated activity. Although business analysts and software developers might not have the skill set or data points to predict revenue streams, they certainly can engage marketing and business folks in project planning and prioritization. Fortunately, there have been many other conversations with folks who reported significant success as they started injecting value-thinking into their feature prioritization processes. I hope you’ll give it a try!

References

JANE CLELAND-HUANG is a professor of software engineering at DePaul University. Contact her at jhuang@cs.depaul.edu.


---

### TABLE 3

<table>
<thead>
<tr>
<th>Delivery sequence</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple signup</td>
<td>−2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−2</td>
</tr>
<tr>
<td>Email dispatch</td>
<td>−3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−3</td>
</tr>
<tr>
<td>Invoicing</td>
<td>−10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Online payments</td>
<td>−5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−5</td>
</tr>
<tr>
<td>Magazine selection</td>
<td></td>
<td>−5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−5</td>
</tr>
<tr>
<td>Magazine edition manage</td>
<td></td>
<td></td>
<td>−5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−5</td>
</tr>
<tr>
<td>Subscription system</td>
<td></td>
<td></td>
<td></td>
<td>−10</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer issue tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>−5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−5</td>
</tr>
</tbody>
</table>

*Each figure represents US $1,000.

---

### TABLE 4

<table>
<thead>
<tr>
<th>Development period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td>−5</td>
<td>−10</td>
<td>−3</td>
<td>−3</td>
<td>−3</td>
<td>−8</td>
<td>4</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>Net cash</td>
<td>−5</td>
<td>−15</td>
<td>−18</td>
<td>−21</td>
<td>−24</td>
<td>−32</td>
<td>−28</td>
<td>−18</td>
<td>−7</td>
<td>5</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>Present value @ 2.50%</td>
<td>−5</td>
<td>−10</td>
<td>−3</td>
<td>−3</td>
<td>−3</td>
<td>−7</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Net present value</td>
<td>−5</td>
<td>−14</td>
<td>−17</td>
<td>−20</td>
<td>−23</td>
<td>−29</td>
<td>−26</td>
<td>−18</td>
<td>−9</td>
<td>0</td>
<td>10</td>
<td>21</td>
</tr>
</tbody>
</table>

*Each figure represents US $1,000.*
WELCOME TO CLOUD TIDBITS! In each issue, I look at a different “tidbit” of technology that I consider unique or eye-catching, and of particular interest to IEEE Cloud Computing readers.

When cloud computing came onto the scene, the benefits of an on-demand, elastic, and efficient place where we could deploy applications became obvious. If you were in healthcare or financial services, perhaps concerns about security and compliance kept you off the cloud. If you were in process control or other real-time applications, perhaps the need to have total control over your infrastructure kept you off the cloud. If you were large enough to already be running your own datacenters, such as the social networks, e-commerce sites, and search engines we use every day, well, your infrastructure essentially became a cloud. And even when you did keep your on-premise servers, you were able to use VMware or Citrix to gain many of the efficiencies that public cloud would deliver. Enterprise virtualization became known as “private cloud.”

Software developers could create their applications however they wanted and, with little effort, could deploy on whatever cloud they chose. Even developers who wanted to use a more specialized application runtime infrastructure, such as Ruby on Rails or Map Reduce, could find multiple providers who would supply that platform on demand. In many cases, these would just be a layer on top of an infrastructure as a service (IaaS) anyhow.

The mode for cloud was always “do your development as you like, and for deployment, clouds are the place to be.” Following this mantra, most developers did indeed easily figure out that cloud platforms—public and private—were in general tremendously compatible. The “atomic particles of IaaS”—compute and storage—although varying by platform, were more similar than different and were available everywhere. A host of scripting languages, such as Chef and Puppet, emerged to automate deployment and management. DevOps engineers could isolate the platform-dependent parts in their scripts, or alternatively use products from the emerging ecosystem to handle deployment and management across several clouds.

Is Amazon Becoming the New Cool Software Company for Developers?

Simple Management Tools and Marketplaces

Cloud vendors began to offer deployment automation options. In 2011, Amazon Web Services came out with AWS CloudFormation and Microsoft came out with AppFabric. Although these were terrific tools, they were very close to the platforms, which had previously been quite austere. The rich
ecosystem of third-party tools was starting to make deployment and management much easier. IaaS was simple, and simple tools went a long way.

Marketplaces and reference virtual appliances supplied all the add-ons anyone needed to round out their applications. Even AWS offered versions of MySQL or Oracle servers, more as a convenience to developers than anything else. IaaS was the common deployment blueprint, cloud platforms were commodity, price competition pushed costs down, and the use of cloud platforms exploded.

Google followed Microsoft’s lead by entering the cloud marketplace with a PaaS-only offering initially, in the form of Google AppEngine. This offering was seen as very limited, and had only cult uptake. Google also eventually released an IaaS offering in the form of Google Cloud Engine, supporting virtual machines directly.

It seemed like PaaSs were relegated for those hopelessly devoted to their codebase, such as .NET on Azure, or Ruby on Rails on Heroku or Engine Yard. Generic PaaS environments such as RedHat’s OpenShift were not catching on. Even AWS’s Elastic Beanstalk had limited success.

When IBM entered the cloud marketplace, it wanted to bring its powerful middleware portfolio with it. Seeing AWS as still pretty much “just a runtime,” IBM and EMC/VMware pushed Cloud Foundry, a new PaaS environment that was supported by many. Cloud Foundry has unique features and capabilities that everyone was quite optimistic about. (Cloud Foundry was covered in a previous “Cloud Tidbits” column.) IBM’s version of Cloud Foundry, IBM BlueMix, included many Websphere and Rational components.

In Search of Value Added

Microsoft had always had a platform-as-a-service (PaaS) strategy in Azure, which was fully expected to provide .NET developers with an easier migration to the cloud. In fact, Azure originally came out as PaaS only, which really didn’t work well for Microsoft, which also saw AWS take huge market share. After losing to AWS year after year, Microsoft was forced to add IaaS. But the .NET-friendly PaaS system and the presence of Microsoft middleware such as SQL servers on the cloud remained as an option, and was used more as a .NET migration strategy than a main strategy. In a symbolic move, Microsoft renamed “Windows Azure” to “Microsoft Azure” and promoted a Java and Linux-friendly environment, but also continued to promote its PaaS environment, hoping it would catch on and provide it the “stickiness” experienced in the past with Windows developers.

Oracle’s approach has been to bring its substantial software development assets to cloud pretty much as is. Oracle calls this a PaaS, even though it’s the same extremely successful middleware the company has had for some time.

The vendors want developers to use their PaaS systems, of course! But up to now, compared to IaaS, PaaS has simply not caught on. Is it because the PaaS systems so far aren’t that revolutionary? Or is it because IaaS is just fine, thank you, largely due to AWS?

Amazon’s AWS answer, revealed during the year and announced at the end of 2014 at its user conference, couldn’t be more interesting.

The Amazon answer has been more interesting. Amazon has clearly decided that being the leader in IaaS isn’t enough. The company also clearly believes that if a PaaS is compelling—forward looking, designed for cloud, and not just “server stuff” repurposed—then it will be compelling. Amazon has decided to become a “software company” and bring forward its own portfolio of middleware (and more importantly DevOps tools) to address this PaaS and tooling innovation gap, even if at the expense of commonality/portability with anyone else.

Amazon’s AWS answer, revealed during the year and announced at the end of 2014 at its user conference, couldn’t be more interesting. Amazon has clearly decided that being the leader in IaaS isn’t enough. The company also clearly believes that if a PaaS is compelling—forward looking, designed for cloud, and not just “server stuff” repurposed—then it will be compelling. Amazon has decided to become a “software company” and bring forward its own portfolio of middleware (and more importantly DevOps tools) to address this PaaS and tooling innovation gap, even if at the expense of commonality/portability with anyone else.

What’s most important to consider here is that AWS introduced a new tool chain, new middleware, and several avant-garde programming models.

The new tool chain consists of CodeCommit, Code Pipeline, and CodeDeploy. With these offerings, Amazon is
hoping that no matter what framework developers are targeting, they use AWS tooling from the git-go (pun intended; CodeCommit includes a Git repository).

The new middleware includes as a centerpiece AWS’s own built-in SQL database, called Aurora. Unlike server versions of MySQL, Oracle, SQL Server, or DR2, which can be found on all the other cloud platforms, this is a cloud native pure database as a service (DBaaS). Aurora is a proprietary technology much like DynamoDB. On the surface, lock-in isn’t a concern because Aurora supports the MySQL syntax. Its highly attractive performance characteristics and pricing are a new competitive threat to legacy database vendors (all of whom are also cloud platform providers). Add this to AWS’s existing cloud native middleware capabilities such as Simple Queuing System (SQS), Simple Workflow Service (SWF), and Simple Notification Service (SNS), and you have a pretty complete middleware offering. Add the direct ability to run code in Elastic Beanstalk and there’s not much missing.

The new programming models are aggressive entrances to the stream-and event-programming fields, with the Amazon Kinesis and AWS Lambda offerings. Amazon Kinesis is a fully managed, cloud-based service for real-time processing of large, distributed datastreams. Amazon Kinesis can continuously capture and store terabytes of data per hour from hundreds of thousands of sources such as website clickstreams, financial transactions, social media feeds, IT logs, and location-tracking events. AWS Lambda is an event-driven compute resource offering, allowing developers to write functions that focus on their core business logic (called lambda functions). These lambda functions can be triggered from many different events from AWS services including Simple Storage Service (S3), Aurora, and Kinesis.

Innovation, but Proprietary
Developers who jump in with both feet now on AWS will find themselves in a proprietary environment. It’s no longer a place where you bring your server images and conveniently use commodity cloud parts to round out a deployment. It looks more like a complete developer environment such as you might expect from incumbents Microsoft, IBM, or Oracle.

Although these incumbents have been trying to tempt developers with their existing offerings, largely repurposed for cloud, Amazon is aiming to lead with innovation, with less and less concern about compatibility or migration than in the past.

IS AMAZON LEAPFROGGING MICROSOFT, IBM, ORACLE, AND EVEN PIVOTAL AS THE NEW DEVELOPER SOFTWARE COMPANY?
Amazon has moved way, way past being a cloud service provider platform. It’s going soup to nuts for developers. If you haven’t thought of Amazon in this way, maybe it’s time to take another look.

And that, my friend, qualified it to be this column’s Cloud Tidbit. I hope you enjoyed it.

Reference

DAVID BERNSTEIN is the managing director of Cloud Strategy Partners, co-founder of the IEEE Cloud Computing Initiative, founding chair of the IEEE P2302 Working Group, and originator and chief architect of the IEEE Intercloud Testbed Project. His research interests include cloud computing, distributed systems, and converged communications. Bernstein was a University of California Regents Scholar with highest honors BS degrees in both mathematics and physics. Contact him at david@cloudstrategypartners.com.
Think Piece

Toward a History of Social Computing: Children, Classrooms, Campuses, and Communities

Joy Rankin
Yale University

During the spring of 1970, Valarie Lamont wrote a computer program to stimulate local environmental activism.1 By deploying text and images in her program to present the history of the Boneyard Creek, which ran through the communities of Champaign and Urbana, Illinois, Lamont created a compelling narrative about the stream’s flooding and pollution problems and potential solutions. As a political science graduate student at the University of Illinois at Urbana–Champaign (UIUC), Lamont investigated citizen participation in community planning, and she produced a program for that purpose. The users of her program—civic leaders, media representatives, local residents, faculty, and students—navigated the narrative by operating the keyboard located below the video screen at their individual terminals. The users gathered information about unfamiliar terms, viewed photographs of the creek and its pollution, expressed their preferences for solutions to the pollution, and provided their comments and opinions to Lamont at any point along the way.2 Lamont and the Boneyard Creek program users worked on a computing system at UIUC known as PLATO (Programmed Logic for Automatic Teaching Operations), a system that by 1970 featured more than 70 terminals on the UIUC campus and at other locations throughout the state.2 Lamont later explained that her programming choice stemmed directly from the community’s existing concern for environmental issues, concern that had been expressed locally and nationally during the first Earth Day events of that same spring.3

Studying Lamont’s Boneyard Creek program illustrates how Lamont and her peers at UIUC employed the PLATO system for personal and social computing, specifically as an activist method to educate citizens about and draw media attention to the problems plaguing the stream. Indeed, Lamont is representative of the individuals I address in my dissertation, “Personal Computing before Personal Computers.” I argue that students and educators using academic time-sharing systems during the 1960s and 1970s transformed computing from a business, military, and scientific endeavor into an intensely personal practice. These time-sharing systems included PLATO, the Dartmouth Time-Sharing System, and several education-centered projects in Minnesota, including the Minnesota Educational Computing Consortium. The users of these systems popularized the now-ubiquitous activity of sitting in front of a keyboard, typing, and responding to messages appearing on a text-oriented display. These students and educators also created communities to support their computing practices, and they fostered social computing.

Rethinking Social Computing

By social computing, I do not mean the recent academic discipline of using software to facilitate social interaction, nor am I referring to particular computing networks such as bulletin board systems (BBSs), the Internet, or Facebook. Rather, I employ the phrase “social computing” to emphasize the social connections forged around and with computing use. We commonly think of computing in individual terms—the lone programmer or hacker, the personal computer, and the user—yet the practice of modern computing has always involved groups of people. From the heterogeneous team of men and women who assembled and programmed the ENIAC computer, to the MIT engineers who gathered around a screen to play Spacewar, to the high school students who programmed PLATO, modern computing has involved the interactions of many people, along with their cultural norms, values, and expectations.

The discipline of the history of computing emerged under the paradigm of the personal computer during the 1970s and 1980s. The Annals of the History of Computing first appeared in 1979. Classic works such as the mathematician Herman Goldstine’s biographical history and the historian Paul Ceruzzi’s Reckoners appeared during this time.4,5 The journalist Steven Levy’s bestseller Hackers cemented the history of computers (at the time) as a history of machines and the great men who built them.6 Indeed, many of us have watched the uptake of the personal computer, have witnessed the cults of personality around Bill Gates and Steve Jobs, and have employed a once-new Internet to our great convenience (or frustration, or both). The history of computing has expanded greatly since then, of course, as historians have examined topics ranging from the history of software to the history of women in computing to the gendering of the computing profession.7 But the personal computer—and the associated elevation of the individual—still drives our discourse.

We historians have only begun to address how people made computing ubiquitous. In an Annals Think Piece...
article published a decade ago, Nathan Ensmenger urged us to a social history of computing—that is, a history of the “many thousands of largely anonymous individuals who contributed to the development of this new social and technological environment.”8 I call for a history of social computing. Moving the adjective highlights the activity of computing as a social and cultural phenomenon. A history of social computing considers how computing has facilitated communication as well as computation. A history of social computing attends to the myriad human interactions that have shaped and supported our digital, networked world.

Why Education Matters
Technologies of education, such as the PLATO system, offer rich opportunities for the study of both personal and social computing. When the researchers at UIUC’s military-defense-oriented Coordinated Science Laboratory initially created the PLATO system as an exploration of the potential uses of computing in education, they had users in mind from the beginning: students. In 1961 Donald Bitzer, Peter Braunfeld, and Wayne Lichtenberger reported on their new computing system, which featured a television screen on which prepared instructional materials were displayed to students as well as keysets (or keyboards) with which students could interact with the instructional materials, including typing responses and seeking additional information.9 Bitzer and his colleagues developed the system with some mental model of “the student,” a point which is not to be overlooked. Valarie Lamont also had some concept of her users in mind when she wrote her Boneyard Creek program. When an individual worked on a system or software, several types of communication occurred: between the individual and the computer, between the individual and her collaborators, and between the individual and her intended user.

When students began using the PLATO system, Bitzer and his team incorporated their feedback and their teachers’ feedback into changes and enhancements to the system. Indeed, I must underscore the methodological value of studying systems that originated in an educational context. Because many of the project publications were oriented toward readers in education, they often included meticulous details of users’ encounters with the terminal, the language, the lessons, the appropriate syntax, and similar issues. These reports documented knowledge and practices that otherwise would have been tacit and unnoticed. Such descriptions are immensely helpful for historians seeking to understand and describe novel computing experiences.

This research draws attention to the important but little studied area of the history of technology in education. The historian of technology Steven Lubar cogently declared, “We have downplayed the skill and knowledge required by users of technology, looking at the machine and not the task, looking for complex systems on the production side, not on the consumption side.”10 Although some historians have begun to address this lacuna by considering household or office technologies, historians are only beginning to study the “skill and knowledge required by users of technology” in schools.11 An exploration of the history of computing in an educational context is particularly promising. In the case of interactive computing systems like PLATO, students and educators were some of the earliest groups of users, and they developed “complex systems” around time-sharing.

Studying technologies in educational settings also means studying children. Now it seems axiomatic that young people and digital technologies simply go together; for example, kids teach their grandparents how to use smartphones.12 What is the history here? How has the relationship between children and technology changed over time? How have technology and the classroom shaped each other? In answering those questions, we historians of computing can engage in fruitful dialogue with others who study media, including radio and television, as well as scholars who consider the history and activity of play. We must grant young people agency as technological actors and study them. Children’s classroom experiences shape both the children and the technology, with what I consider an accretion of technological exposure. For the K-12 students in New Hampshire, Minnesota, Illinois, and elsewhere who used time-sharing in the 1960s and 1970s, those early computing experiences were formative. Bill Gates had his first computing experience on a time-sharing system.13 Moreover, students and educators were not simply consumers of time-sharing. My research demonstrates that they generated new knowledge about this form of computing, including writing numerous software programs and devising modes of communication and resource management.

Examining educational technologies also means considering their social setting,
whether the classroom, campus, or community. Indeed, although various forms of computing, ranging from mainframe computing to time-sharing to mini-computing, proliferated on university campuses during the 1960s through the 1970s, we know little about how individuals and groups used and responded to those computers. Most historians have depicted the campus protests of the 1960s as a rejection of technology and technocracy. A closer examination paints a different picture. Although Bitzer and his colleagues developed PLATO for military research, Lamont later deployed it to stimulate environmental activism. While Lamont was writing the PLATO Boneyard program, hundreds of UIUC students protested the installation of the Illiac IV computer on campus after they learned that the Department of Defense controlled most of the computer time. Shortly thereafter, on 2 March 1970, hundreds of students protested General Electric’s on-campus recruiting, and the university administration called in the National Guard to enforce a curfew.

For many UIUC students, PLATO represented personal computing and democracy. At the same time, Illiac IV symbolized the evils of the military-industrial complex and the Cold War. To better understand the nuances of American activism, protest, and politics during the 1960s and 1970s, and to understand the environments in which personal computing emerged, we must carefully attend to these contingencies.

References and Notes


2. V. Lamont, “New Directions for the Teaching Computer: Citizen Participation in Community Planning,” Computer-Based Education Research Laboratory, Univ. of Illinois, Urbana, July 1972, CBI Archives, collection 133, box 5, folder 72jn02.


14. Here, I am focusing on the majority of people on university campuses (students, faculty, and administrators) who were computing. Histories of computing on campus tend to address particular projects and the individuals who worked on those projects, such as MIT’s Multics or Engelbart’s work at the Stanford Research Institute. See, for example, Levy, Hackers; A. Norberg and J. O’Neill, Transforming Computer Technology: Information Processing for the Pentagon, 1962–1986, Johns Hopkins Univ. Press, 1996; and T. Bardini, Bootstrapping: Douglas Engelbart, Coevolution, and the Origins of Personal Computing, Stanford Univ. Press, 2000.


Joy Rankin is a doctoral candidate in history at Yale University. Contact her at joy.rankin@yale.edu.
Nate Fick, CEO of Endgame, discusses cybersecurity, the term cyberwar from a Marine’s perspective, and his time at the Center for a New American Security.

Many of us in computer security have no idea what real war is like, so we blithely throw around the term cyberwar. Are we watering down reality way too much?

I see a bright red line between the kinetic and nonkinetic worlds, and philosophically, I learned my lessons in the kinetic world. In the kinetic world, if you’re going to kick a hornet’s nest, you better be sure you’ll kill all the hornets. In the cyber domain, for a whole bunch of structural reasons, I don’t think that’s possible. We need to be very careful in how we talk about offensive cyber capability and cyberwar. Another bright red line for me is between the federal space and the commercial. They’re wildly different conversations, and too often we conflate them.

I loved your book, One Bullet Away. What made you write it?

The book’s dedicated to my good friend and comrade, Brent Morel. Brent was my hand-picked replacement; he was killed on April 7, 2004, leading the platoon in Anbar Province. I decided to write about the experience after Brent was killed because I felt that we had heard from generals, journalists, and politicians, but we hadn’t heard much from the guys who were actually fighting the war.

It’s good for people to understand the realities of what you all went through for us. It feels somewhat trite, but I want to know the answer: How’s leading a start-up similar to and different from leading a combat platoon?

It isn’t trite at all. The combat and start-up worlds are both characterized by euphoria and terror in rapid succession. I think I got spoiled in the Marines because people actually did what I said, and in the start-up world, that’s not always true. But I think they have a lot of similarities. I had the privilege of working for some great leaders in the Marine Corps, and the best of them was a guy who, if he said, “Go up on the roof, do a swan dive, and figure out how to stick the landing on your way down,” I would have saluted and said, “Aye, aye, sir,” and gone and done it. I once asked him about his leadership philosophy—why we all followed him through the gates of hell and were happy to do so. He said, “I’ll give it to you in three words, Nate: officers eat last.” For him, leadership wasn’t about privilege, it was about responsibility. What did that mean day to day?

He told us what to do, but he didn’t tell us how to do it. I learned a lot from him, so even though I don’t set up machine gun positions or plan ambush patrols anymore, a lot of the intangible stuff I learned in the Marines I still draw on every single day.

After that, you joined the Center for a New American Security as one of its founding fellows, eventually becoming CEO of that organization. What does CNAS do, and why is it important?

It’s a nonpartisan research organization. Many people would be surprised to learn how much
About Nate Fick

Nate Fick is CEO of Endgame, a security intelligence and analytics company. He has more than a decade of experience in the security community, and is an operating partner at Bessemer Venture Partners, where he focuses on defense and intelligence technologies. Before joining Endgame, Fick was CEO of the Center for a New American Security, a national security research organization. He also served as a Marine Corps infantry officer, including combat tours in Afghanistan and Iraq. His book about that experience, One Bullet Away, was a New York Times bestseller, a Washington Post “Best Book of the Year,” and one of the Military Times’ “Best Military Books of the Decade.” Fick graduated with high honors in classics from Dartmouth College and holds an MPA from the Harvard Kennedy School and an MBA from the Harvard Business School.

cutting-edge thinking, innovation, and policy formulation is done outside the government, where people aren’t living their days driven by the tyranny of the inbox, and you can challenge conventional wisdom openly without worrying about the ramifications.

CNAS was started by Michèle Flournoy and Kurt Campbell in 2007 to bring together a critical mass of national security thinkers who weren’t aligned with any one political party or in the pocket of industry. Every project we ever did was signed by individual people; the organization never took an institutional position. It’s a young organization filled with young people—not necessarily young in terms of age, but fresh in their thinking and willing to challenge conventional wisdom. In a field where gray beards dominate, where your credibility often moves in lockstep with your age, we wanted to break that mold and build a different kind of think tank.

The national security establishment seems enamored with cyber offense but is seriously confused about what constitutes defense. What should cybersecurity defense look like?

I think it starts with the mantra, build better software. If there were to be a silver bullet, that’s as close as we would get to it, right? Beyond that, I think we have a skillset imbalance in the sense that the ability to do harm on the offensive side is becoming increasingly commoditized. The barriers to entry are coming down around the world, and you see it in the marketplace, in companies operating in that space suddenly competing with Romanian teenagers. But ultimately, I think we’d all be better served if we could make defense cool in the same way that offense has this aura about it.

Even if we scrutinize the Snowden documents, it’s very clear that the attribution problem on the Internet hasn’t been solved. Why does knowing exactly who’s attacking you matter?

It matters if you’re going to fight back. If you’re only defending yourself, it doesn’t matter. If we’re talking about building better armor on our tanks, it doesn’t really matter where the inbound shells are coming from. If we’re talking about missile defense, the strategic defense initiative, and everything that flowed from it, you don’t necessarily need to know where the shot came from. If you’re going to retaliate, if you’re going to fight back, then attribution becomes essential.

I’m always skeptical when somebody says, “This time is different,” because you know what? This time is never different. On the war fighter side, as soon as you’re talking about firing back, you need to abide by a couple of principles that are intrinsic to how we as Americans fight and to how, I would argue responsible, forces have fought all the way back to Thomas Aquinas, the four from whom these two principles flow. The first of them is noncombatant immunity. If you’re going to fight back as a state using sanctioned power, where the state has a monopoly on the legitimate use of force, you have to do everything possible to make sure that you’re not impacting noncombatants. The second is that your response must be proportional. If somebody punches me in a crowd in Iraq, I can’t reply by firing my M4 at him, right? A sense of proportionality and a sacred tenet of noncombatant immunity—let’s acknowledge those and then we can have the conversation on how achievable those things are in the cyber realm and what that looks like.

We ask the NSA to spy, which it’s exceptionally great at, but we also ask it to secure our cybersystems, which makes spying more difficult. We’re asking these guys, our own people, to work at cross-purposes, which is basically impossible. What’s a better solution?

I can agree with you on the problem statement, but I’m not sure I have a better answer. Go back 30 or 40 years, and you could actually target collection outside the US. Remember during the Cold War, when the US had submarines tapping undersea cables? We could actually access data—conversations—that could only be happening outside the US. But with the global distribution of software and the way that our global communications architecture has evolved, the world, technology, and norms have changed faster than law and policy.
Our government has to pick up the pace if it’s going to work constructively with technologists and the private sector. Maybe we need a cyber-focused, federally funded research and development corporation, something like what RAND in Santa Monica, California, was for the nuclear era. Imagine something focused on cyber that doesn’t require a classified staff—you can have a very diverse group of people, and you don’t have to wear a dark suit to work. It could be an excellent way to try to improve the tidal ebb and flow between the two worlds, because right now it’s just not good enough.

Why are liberal arts important for business and military leaders?
The basic tenets of leadership don’t really change, and I think that one of the most important things you can do as a military officer or as a CEO is to explain and contextualize. Build shared contexts, set a vision, set a direction, build the context so everybody understands it, and then get the troops marching in the same direction. I spend a lot of time feeling as if I’m steering a wooden boat down a river through the rapids, and I try to swing the rudder enough to avoid the big rocks, but every now and then, we hit one, and a couple people fall out of the boat, and you’ve got to pull them back in and patch the hole, and keep everybody aboard until you hit the next rock, and you do it all over again. War feels a little bit like that. I think growing a business feels a little bit like that. That’s a fundamentally human thing, right? It requires connecting with, understanding, and communicating with people in speaking and in writing. These aren’t technical skills. Do I wish I could layer an engineering degree in there for myself? Absolutely. I spend a lot of time with our guys trying to better learn the intricacies of what they do, but the reality is it’s not how I spend my time, and I don’t think it should be. I think we need leaders who are building context and helping their subject matter experts have the direction, the resources, and the support they need to do their jobs well.

The Silver Bullet Podcast with Gary McGraw is cosponsored by Cigital and this magazine and is syndicated by SearchSecurity.

Gary McGraw is Cigital’s chief technology officer. He’s the author of Software Security: Building Security In (Addison-Wesley 2006) and eight other books. McGraw has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.

The Human Intranet—Where Swarms and Humans Meet

Jan M. Rabaey, University of California at Berkeley

EDITOR’S INTRO

Over the last few years, mobile phones have evolved from being simple communications devices to powerful computing devices that let users access to the Internet whenever they want. In the next phase of development of the mobile Internet, we’ll see a large number of interconnected devices, such as thermostats, cars, and home appliances. This complex system of devices and networks is what Jan Rabaey calls a swarm. These new devices will be interconnected in complex ways, and the users will consume information in novel ways from the swarm, from a variety of different devices rather than a single device. Rabaey presents a compelling vision for the future of connected devices.

—Nayeem Islam

The Human Intranet represents a natural evolution of the smartphone into a system that features a far broader and richer set of interface modalities. This truly transforming technology will not only change how we interact with our environment but also how we observe, operate, and extend ourselves. It all starts with concepts that find their roots in the Internet of Things (IoT) and swarm technologies.

THE IoT AND SWARMS

There’s no question about it—IoT is happening as we speak and is radically transforming the information technology platform. In the last decade or so, the cloud has emerged as the keeper, transformer, and interpreter of all data, and mobile devices, such as smartphones, have changed how we enter, access, and interact with information. The IoT adds yet another layer to the onion, providing an extremely high-bandwidth channel between the cyberworld (represented by the cloud) and the physical and biological world in which we live (see Figure 1), giving birth to terms such as cyberphysical and cyberbiological systems. For the first time, we can engineer systems that tightly interweave the “real” physical and “imaginary” cyberworlds, often blurring the boundary between the two.

To imagine how interwoven the two might become, consider the projection made in 2006 that by 2017, there would be 7 trillion wireless devices serving 7 billion people, which is equivalent to 1,000 sensors per living person! This projection may seem outrageous, but the current progression of sensor deployment and its forward projection, as illustrated in Figure 2, shows that while we may not be entirely on track in reaching that goal, we may very well reach it in the mid 2020s.

Much has been written about the possible effects and applications of the IoT, covering virtually every aspect of society: industrial and home automation, mobility, energy and the environment, agriculture, safety and security, health and wellness, art, and social interaction. However, the nature of many of the applications envisioned is hampered by the “IoT” name itself. It conjures an image of many devices connected through a vast network to a “centralized” cloud that acquires and acts on that data. Although this picture might work well for some functions, it misses a great number of scenarios and could act as a hurdle for adopting other ones.

Instead, imagine a world permeated with connected smart devices with sensory, actuation, compute, and storage capabilities. Some might be static, while others might move around rapidly—such as those carried by humans or mounted on cars or drones. In such an environment, applications would form by opportunistically marshaling the resources available to them at a given time and place. Such a distributed system is called a swarm, a term that captures the organic nature of cyberphysical and cyberbiological applications better than the Internet-centric IoT concept.

While swarms might consist entirely of non-biological entities (such as clusters of cars on the freeway, or bands of drones in the sky), often they intimately involve one or more humans. In fact, some of the highest-impact uses of the sensory swarm might relate to how humans interact with the physical world around them (and the cyberworld beyond), how they interact with their fellow human beings, and...
ultimately how they monitor and introspect themselves.

SWARM CONCEPTS AND TECHNOLOGIES

The IoT and swarm concepts aren’t new by any means. They find their origins in the joint ideas of ubiquitous computing (1980s) and sensor nets (1990s), which, despite their initial appeal, failed to make sizable impact and were confined to the fringes of the information technology world for a long time. A broad range of arguments can be fielded to explain why this occurred: immaturity in low-energy and wireless technologies; the inherent complexity of distributed systems; and the lack of standards, programming interfaces, robustness, and ease of use. The most important reason, however, is probably a missing operational model that could lead to “economy of scale” deployments.

Most sensor-net applications were developed as stovepipes, addressing only a single application space and resulting in incompatible proprietary vertical chains. A perfect example can be found in the smart building space, in which separate technologies were developed for environmental monitoring, energy management, automation, and security—but never for all at the same time.

Much has changed in the past few years. Ubiquitous wireless connectivity is emerging, and a broad range of low-energy wireless transceivers are readily available and have been integrated into both access points and smartphones (near-field communication and Bluetooth LE being the latest). Also, low-energy embedded processing platforms, such as Arduino and Raspberry-Pi, are ubiquitous, and multisensor programmable modules are available from a range of vendors. Combine this with the fast prototyping capability offered by 3D printers, and it’s no surprise that a slew of creative swarm (IoT) devices are being spawn at a breathtaking pace (Figure 1, 2). And the Internet of Things (IoT) and swarm technologies.

The Human Intranet—Where our environment and cyberspace meet

The Human Intranet represents a new computing paradigm that tightly interweave the “real” world and the “imaginary” cyberworld (represented by the cloud), capturing the organic nature of cyberphysical interactions. This system is called a swarm system, as it is an organic network that self-organizes and adapts to changing conditions.

Many devices will be interconnected in complex ways, and the users will interact with our environment but also with external systems. For the first time, the proliferation of connected smart devices with sensory, actuation, compute, and storage capabilities, along with the resources available to them at a given time and place, such as mobile networks and powerful computing devices that let users access to the Internet, comes together to create a new computing paradigm, the Human Intranet (Figure 2). These devices will be interconnected in complex ways, and the users will interact with our environment but also with external systems. For the first time, the proliferation of connected smart devices with sensory, actuation, compute, and storage capabilities, along with the resources available to them at a given time and place, such as mobile networks and powerful computing devices that let users access to the Internet, comes together to create a new computing paradigm, the Human Intranet (Figure 2).

Figure 1. The evolving information technology platform. Over the past decades, information technology has been moving to a concentric model of centralized servers (the cloud) communicating with ubiquitously distributed mobile access devices. The emergence of IoT is adding an additional layer of devices to that picture.

Figure 2. Actual and projected growth of sensor deployment based on the predictions from a number of leading research labs or companies. (Image courtesy of Janusz Bryzek, Fairchild and chair of TSensors Summit; used with permission.)
pace (consider, for example, the innovative concepts emerging from the Citris Invention Lab; http://invent.citris-uc.org, as well as various other maker labs).

While the hardware platform is maturing, the supporting software environment is lagging. Innovative programming environments are being envisioned, ontologies constructed, and APIs proposed. Yet most proposed IoT software platforms are cloud-centric and thus fail to meet the stringent latency, energy-efficiency, robustness, and privacy requirements essential to the swarm concept. At the Berkeley Ubiquitous SwarmLab (http://swarmlab.eecs.berkeley.edu), we’ve been developing the SwarmOS as an alternative—an open and universal platform to foster the creation of a broad range of innovative swarm applications (see Figure 3).4

The development of such a distributed operating system faces many challenges, but one of the toughest is guaranteeing service—that is, ensuring that an application performs reliably and predictably under all possible circumstances, even while sharing the platform among many applications. This stands in stark contrast with the Internet model, which is purely functional and—by design—doesn’t define or impose performance metrics. In the swarm space, a failure to address timelessness constraints can be catastrophic or life threatening. Consequently, at the core of the SwarmOS is a “brokerage function,” which dynamically trades between the needs of the running applications (the “swarmlets”) and the availability of resources, with the mechanism of micro-payments as the means to ensure balance and fairness.

**SWARMS AND HUMANS**

While the proliferation of communication and data processing devices (such as laptops and smartphones) has profoundly changed our interaction patterns, nothing has similarly changed our means of processing inputs (sensory) and outputs (actuation). Many of these interactions are still funneled through a limited set of means (such as displays, headphones, keyboards, touch panels) integrated in a single device. The swarm could change all of this.

Consider the evolution of the smartphone. Over the past decade, it has continuously been accumulating additional functionality in terms of connectivity options and sensory capabilities. Yet trying to integrate all of these into a single device limits both the user experience as well as the application scope. For example, many meaningful signals, such as ECG, are impossible to acquire in a single handheld device. This motivates the various efforts on disaggregating the phone into an ensemble set of separate but connected components, such as watches, bangles, glasses, contact lenses, earpods, and other wearable devices.

In this scenario, which researchers at the Berkeley Wireless Research Center have dubbed the unPad (see https://bwrc.eecs.berkeley.edu/research/unpad-and-ewallpaper), the personal communication device is no longer a single entity; it becomes a collection of devices aggregated in true swarm fashion in an organic and opportunistic way. Some of those components can be carried on the person, while others might be provided by the augmented environment around us. Given the broad diversity of sensory and actuation interfaces offered by advanced technology, these unPads would offer an experience that’s substantially richer than what our five natural senses and traditional motor functions (speech, motion) can offer. The potential is huge—think empowered humans in an enhanced world. The nascent field of brain-machine interfaces offers just a glimpse into what’s possible.5

Realizing this potential requires overcoming a number of barriers, similar to those the swarm is facing but even more challenging—not only technologically but also in terms of the economical and sociological aspects.

**THE HUMAN INTRANET**

The transformational opportunity offered by wearable devices has certainly not escaped the industry—or the press.6 Yet the technological solutions being forwarded today bear an eerie
resemblance to last decade’s sensor-net scenario. Many devices are single-purpose gadgets connecting in a point-to-point link to a smartphone and are only compatible with devices of the same company—again, a true stovepipe model.

Imagine, in contrast, a Human Intranet realized as an open, scalable platform that seamlessly integrates an ever-increasing number of sensor, actuation, computation, storage, communication, and energy nodes located on, in, or around the human body acting in symbiosis with the functions provided by the body itself. The traditional set of senses and interactions is to be augmented by a set of new capabilities, some of which might be hard to even imagine today. Added functionality might be extrospective—that is, dealing with the external world around us—or introspective—including monitoring, intervention, interaction, and augmentation of human operation.

Like the swarm, the Intranet develops organically as a heterogeneous mesh of connected nodes (wired or wireless), collaborating to deliver services in a guaranteed way, notwithstanding the stringent environmental, energy, and size constraints.

Again, the vision of body-area networks is certainly not new. Yet the vast majority of the proposed approaches rely either on point-to-point connections or dedicated star networks serving only a single purpose. For the transformative potential to be realized, we need to take a fresh look and harness Metcalf’s law. For this, devices must be seamlessly “insertable” and interfaces must be seamlessly combinable, all while the interaction with the environment is maintained independent of the available connectivity options.

In addition, given the personal nature of the information being acquired and transmitted, as well as the potentially life-threatening effects of indiscriminate or malicious actuation/stimulation, any solution should provide rock-solid reliability, safety, and security guarantees, in stark contrast to current practice.

To get an idea of what an Intranet might look like, consider the neuro-prosthesis system shown in Figure 4. A network of sensors measure neural activity as related to motor function (using either on-skull EEG or implanted neuro-electrodes). The information is transmitted through a combination of wireless and wired connections to one or more control modules (“hubs”), which translate the intent into actual control signals driving an exoskeleton or a prosthetic device. Those hubs, an evolved version of the smartphone, also support telemetry and direct interfacing with the surrounding environment. Additional sensors could measure EMG signals at various muscle groups or collect tactile feedback from the exoskeleton.

Figure 4. An example Human Intranet configuration targeting a neuro-prosthesis application. An exoskeleton is operated based on information obtained from distributed sensors acquiring neural and other biometric signals. A body-spanning network distributes both information and energy.
This system features many of the properties that are typical for Human Intranet applications:

- It’s distributed over the entire body.
- It integrates a collection of diverse devices including sensing, actuation, processing, and storage.
- It combines energy-starved (batteryless) devices with energy-rich battery-operated nodes (for example, an evolved smartphone).
- It exploits a broad range of communication strategies (both wired and wireless), not only for information but also for energy delivery.
- It provides high-capacity, low-latency connectivity to the surrounding augmented environment.
- It must be continuously operational for extended periods of time—although the amount of activity may vary dynamically over time.

Many similar scenarios can be envisioned including humans immersed in virtual reality or augmented environments.

**INTRANET CHALLENGES**

To realize the Human Intranet, a number of technological challenges need to be overcome, some of which I discuss here.

**Integrate Energy and Information Distribution**

Energy sparsity is one of the central challenges in constructing the Human Intranet. While some nodes (hubs) might have a sizable energy reservoir in the form of batteries or energy-harvesting capability, others might not have any storage and thus would require remotely provided energy when information is requested.

Paralleling, in a sense, the human nervous and arterial systems, network nodes collaborate to form a hierarchical and adaptive mesh that delivers both information and energy. Network links exploit a broad range of connectivity mechanisms, including wired and wireless, electromagnetic, resistive, capacitive, inductive, acoustic, and optical—the choice of which is determined by the local context.

The evolved smartphone plays an essential role in this scenario as a hub node. It serves as a bridge to the surrounding world with a diverse set of broadband communication capabilities (5G and beyond). While providing major computation and data storage capacity, it also serves as an energy reservoir for the rest of the Intranet. Devoid of many of its user interface functions of today, its form factor could become really small, rendering it unobtrusive. A believable model of such a hub is presented in a series of videos, called “A Day Made of Glass,” created by Corning Glass (see www.corning.com/ADayMadeofGlass/Videos/index.aspx).

**Distribute System Intelligence**

The Human Intranet operates in a dynamic world, subject to both slow evolution and extremely fast changes in needs, activity, conditions, and composition—both in the surrounding environment and in the Intranet itself. Therefore, the Human Intranet should be constructed as an adaptive and evolutionary system that combines local decision making with centralized global learning and optimization performed in hub nodes. This approach, in which intelligence is both global and distributed, is essential to address issues of latency and single points of failure, while avoiding the trap of many distributed entities with limited knowledge trying to address a global issue.

**Ensure Fail-Safe Operation**

Given the often life-critical nature of its applications, basic or partial functionality of the Human Intranet must be retained under all circumstances, even when resources fail or are insufficient, during system overload, or during denial-of-service attacks. Fail-safety must be built-in from the ground up and should be an inherent property of the basic components and their compositions. Approaches to address this include relying on baselining and safe modes, exploiting redundancy, and using inherently adaptive and reconﬁgurable network strategies.

**Develop a Human Firewall**

Given the personal nature of the information being acquired and transmitted as well as the potentially life-threatening effects of indiscriminate or malicious actuation/stimulation, any solution should provide rock-solid safety and security guarantees. At Berkeley, we envision a combined and integrated set of mechanisms—such as unique biomarkers, mandatory encryption, and adaptive cloaking—jointly labeled the Human Firewall—working to ensure that private data circulating in the network remains secure and the network is protected from external intrusions.

**ACKNOWLEDGMENTS**

This article is the result of many discussions with a broad range of people—in particular, my colleagues at the Berkeley Wireless Research Center (Elad Alon, Ana Arias, Robert. Brodersen, Ali Niknejad, Borivoje Nikolic, Vladimir Stojanovic, John Wawrzynek, and Paul. Wright), the Berkeley Ubiquitous Swarm Lab (Bernhard Boser, Bjorn Hartman, Edward Lee, John Kubiatowicz, Eric Paulos, Kris Pister, Claire Tomlin), and the Center for Neural Engineering and Prosthetics (Jose Carmena and Michel Maharbiz). The gracious support of the member companies and funding sources of these centers is gratefully acknowledged.
REFERENCES


Jan Rabaey holds the Donald O. Pederson Distinguished Professorship at the University of California, Berkeley, where he is also scientific co-director of the Berkeley Wireless Research Center (BWRC) and founding director of the Berkeley Ubiquitous SwarmLab. His current interests include the conception and implementation of next-generation integrated wireless systems over a very broad range of applications, as well as exploring the interaction between the cyber and the biological world. Contact him at jan_rabaey@berkeley.edu.
Surface manipulation and representation is becoming increasingly important, with applications ranging from special effects for films and videogames to digital fabrication and architectural design. Despite significant research efforts, there is still a large technological gap between the acquisition of 3D models and the tools used to process, edit, and render them. Acquired 3D objects do not possess a high-level structure; they often start as a “point cloud” of surface points from which a triangle mesh is derived (see Figure 1). Conversely, the majority of the 3D models used in movies, games, and CAD applications are smooth surfaces with a shape that is controlled by a coarse grid of quadrilaterals. Depending on the application, the quadrilaterals can be either subdivided or used as control points for defining high-order polynomial surfaces.

The conversion of a triangle mesh into a quadrilateral mesh is a difficult problem that has been deeply studied during the last decade. Many different approaches have been proposed and transferred from academia to commercial modeling and CAD software. However, there are still many open problems to solve to provide a fully automatic pipeline that converts an unstructured model into a high-level representation that can be directly used in a conventional 3D modeling pipeline.

This article explains why it is important to represent surfaces using quadrilateral meshes, while providing an overview of the conversion techniques that I invented to retopologize triangular meshes. (For a complete literature overview, interested readers are referred to the quadrilateral meshing survey written by Bommes and his colleagues.) After showing how these algorithms can be applied to contemporary problems in architectural geometry, their limitations and interesting future research directions are discussed.

Quadrilateral Remeshing

The creation of triangle meshes has been extensively studied in the last 30 years, and many robust algorithms are currently available in production-quality software libraries. A triangle mesh is the preferred choice for real-time rendering because it is well-suited for the classic rasterization-based rendering pipeline used by modern GPUs. Its main limitation is that it does not contain any global structure, and it is thus difficult to edit or use as a control grid for higher-order surfaces. This is why quadrilateral meshes are ubiquitous in the movie industry and in CAD applications: they naturally represent a pair of directions, and they exhibit global structures called edge loops (see Figure 2), which are chains of edges that can be selected by...
picking any edge in the loop, increasing the efficiency of modelers, riggers, and animators.

The de facto approach to model smooth surfaces in CG movies is the Catmull-Clark subdivision, which is a recursive algorithm that produces a smooth surface with a shape that is completely determined by a coarse control grid. Finally, quadrilateral meshes are often preferred over triangular meshes for discretizing partial differential equations (PDEs) because of their numerical properties.

**Mesh Quality**

The quality of a triangular mesh is usually determined by the shape and size of the triangles; equilateral and uniformly sized triangles are often preferred for rendering smooth surfaces, where they provide an even interpolation of the per-vertex normals. High-quality triangular meshes also exhibit better numerical properties when used for solving partial differential equations.

Although similar quality metrics could be defined for quadrilateral meshes, they are not useful quality measures because quadrilateral meshes are not directly used to represent 3D shapes. A quadrilateral mesh is usually used as a control grid for a subdivision or NURBS surface (Figure 1). In both cases, the quality of the final surface is mainly affected by the distribution of singularities that are the vertices touched by more or less than four edges. These vertices are particularly important because they are the only ones where the quadrilateral mesh is not a regular grid. The limit surface of a Catmull-Clark subdivision is only $C^1$ smooth on singular points, and it is challenging to smoothly connect more than four NURBS that meet at a single point.

A singular vertex can have a different singularity index, depending on the number of edges touching it, and the total number and index of the singularities is a topological invariant that depends on the genus of the surface.

The quality of a quadrilateral mesh is thus measured by the following:

- the number of singularities used (the lower the better);
- the geometric location of the singularities (the generated artifacts in the smooth surface should be hidden or not disturbing); and
- the alignment of the edges, aka edge flow (edges should be aligned with meaningful features, thus allowing artists to quickly select semantically meaningful edge loops).

The last two requirements are difficult to evaluate objectively and are difficult to incorporate in an optimization algorithm. Many quadrilateral meshing algorithms incorporate user input to allow users to adjust the result depending on their preferences.

**Computer-Assisted Quadrilateral Remeshing**

Traditional modeling tools allow us to retopologize a surface by manually placing its vertices and edges. This procedure is time consuming and prone to errors; because the number and placement of singularities is a global feature of the mesh, it cannot be modified using local operations such as insertion of vertices and edges. Thus, if a mistake is made while retopologizing a part of a shape, it is not possible to fix it locally, and the entire model has to be retopologized from scratch. Surprisingly, this procedure is still the main method used by artists and CAD engineers to design coarse quadrilateral control grids, and computer-assisted quadrilateral remeshing methods are only recently starting to be used instead.
There are three major families of quadrilateral meshing algorithms:

- **Local methods** adapt simplification strategies developed for triangular meshes to quads. Although they tend to introduce a large number of singularities, their efficiency, simplicity, and stability make them ideal for processing large and noisy datasets.
- **Global methods** use global optimization strategies to optimize directly the number and placement of the singularities. They generate high-quality results, at the expense of efficiency and controlability.
- **Interactive methods** integrate local optimization algorithms with smart user interfaces, providing semiautomatic tools that are both efficient and highly controllable.

**Local methods.** I proposed two local methods to convert a triangle mesh into a coarse quadrilateral mesh. They first trivially pair every two triangles in a quadrilateral, thus producing a dense quadrilateral mesh, and then reduce the number of quads using simple and efficient local operations. Each local operation affects a small part of the quadrangulation, reducing the number of quads by collapsing one of them or improving the shape of the quads by rotating an edge (see Figure 3).

Although this technique does not provide explicit control over the placement and number of introduced singularities, it was the first completely automatic method able to create coarse quadrilateral meshes.

The algorithm is simple, robust, and as opposed to some of the methods presented in the next sections, it can be applied to dense meshes. The 3D positions of the quadrilateral layout can be optimized so that the induced smooth surface (obtained with Catmull-Clark subdivision) is as close as possible to the original geometry, whose details can be stored as a displacement map (see Figure 4). Note that the created mesh is adaptive—that is, it concentrates the elements in the regions with more geometric details. This feature can be disabled for applications that benefit from uniform elements.
Global methods. Global methods rely on a global optimization to decide the number and position of the singularities. In this section, I focus on my algorithm that generalizes previous work by allowing adaptive and anisotropic quadrilateral meshes.5 Figure 5 illustrates the algorithm’s five major steps:

1. Edge alignment constraints are manually defined by a user or extracted from principal curvature directions. The user-specified constraints are a sparse set of quadrilaterals, with size and edge alignment that will be preserved in the final quadrilateral mesh and interpolated in the remaining regions.

2. The constraints are interpolated over the surface, defining a set of directions over the entire surface. Different algorithms can be used for this task, each providing different control over the type of constraints that the user can specify.5,6

3. The shape is deformed depending on the interpolated constraints, enlarging regions where a higher density of quadrilateral has been prescribed.5

4. The deformed directions are used as gradients for a global parametrization, whose principal axes are aligned with the specified directions in the least-square sense. The integer isolines of the parametrization naturally define a uniform quadrilateral mesh on the surface. David Bommes and his colleagues introduced this method in 2009,7 and many variants and extensions of this algorithm have been proposed in the last four years.8

5. The deformation (step 2) is reverted, revealing the final quadrangulation that follows the constraints specified by the user in the first step.

The major advantage of global methods is that they provide precise control over the singularities of the resulting quadrilateral mesh that can be either prescribed manually or automatically optimized (see Figure 6). Their main disadvantages are the extreme implementation complexity and that they might fail to generate a pure quadrilateral mesh. Depending on the constraints, these methods might introduce triangles or pentagons close to singularities.

Step 2 can be further enhanced by extracting symmetries from the original object and enforcing them during the interpolation of the alignment constraints to create a symmetric quadrilateral mesh.9 This is particularly useful for man-made CAD objects that often exhibit multiple bilateral symmetries (see Figure 7).

Interactive Methods. Exact control of every edge of the quadrangulation is required in certain applications, such as the creation of extremely low-poly videogame characters or the creation of high-end movie characters. In these cases, fully automatic algorithms are not applicable because they do not provide sufficient control. The current solution is to manually draw every quad, a task that can consume multiple days of manual labor on complex models.

To tackle this problem, I introduced a new interaction paradigm10 to efficiently quadrangulate a surface using a sketch-based interface, while having accurate control over each vertex and edge wherever necessary. In contrast to the parametrization-based approaches, the users draw a subset of the edges they want to have in the final mesh, and the system takes care of filling the regions between the strokes with quadrilateral elements.

The basic mode allows us to freely draw sketches on a surface: when a loop is formed, the region enclosed by the loop is tessellated (see Figure 8). For every region, it is possible to change the number of edges on each of its sides and change the connectivity in its interior.

Figure 5. Five major steps of a global, parametrization-based, anisotropic meshing algorithm, from left to right. (a) Constraints (green quads) are provided by an artist, and (b) they are converted into a frame field, (c) which is deformed together with the shape until it becomes a cross field. (d) The deformed mesh is quadrangulated using a global parametrization approach and (e) then warped back to its original shape.
To further improve the efficiency, the system provides a set of tools for the most common cases. A brush tool can be used to quickly create a strip of quadrilaterals, the autocompletion tool automatically fills the region below the cursor, and the cylinder tool creates a circular pattern around cylindrical regions such as fingers and limbs.

**Impact and Applications**

Automatic and assisted remeshing algorithms are now available in commercial modeling softwares such as ZBrush and 3DCoat. They are daily used by hundreds of artists and engineers to create coarse and dense quadrilateral grids for high-end videogames, movie visual effects, and engineering applications. Quadrilateral meshing algorithms are also deeply affecting architectural designs, enabling architects to design and build free-form structures that were impossible to realize with traditional techniques.

**Planar Panelization**

Glass and steel constructions are common because of their combination of functionality and specific aesthetic. However, their construction is expensive if curved glass panels are used. Tessellating a surface with flat quadrilaterals is a challenging problem that requires expensive numerical optimization.

With slight variations to the algorithms presented earlier, it is possible to tessellate a surface with planar quadrilaterals. Starting from a planar tessellation, a building can be constructed by replacing each face with a flat glass panel that is much less expensive to manufacture.
cally, the edges of a quadrilateral mesh with flat faces define a conjugate field, which is challenging to design. I recently proposed a simple and efficient algorithm that allows architects to quickly experiment with different planar tessellations by simply specifying a set of desired alignment constraints (see Figure 9).

**Free-Form Masonry Structures**

Another way in which quadrangulation could be used is the design and tessellation of free-form masonry structures. These structures consist of unsupported stone blocks and stand thanks to their special geometry where all blocks are in static equilibrium. Masonry structures have been widely used in the past by following simple patterns for the construction of arcs and vaults, and only recently they are being used in modern architecture to realize free-form designs.

I proposed a completely automatic pipeline that converts a sketch of a surface into a masonry model consisting of hexagonal blocks. An automatic quadrilateral remeshing algorithm creates a coarse force layout that is used to transform the given sketch into a self-supporting surface (see Figure 10). The same algorithm is then used to split the surface into a staggered grid of blocks. The block pattern is a quadrilateral mesh, where specific edges are removed to create a staggering effect that increases the interlocking between the pieces, simplifying the construction and improving the structural properties of the masonry building. We validated our approach by fabricating and assembling small-scale models of masonry buildings.

**Open Problems**

Quadrilateral meshing is still an open problem with many challenges left to address. The plethora of new applications that use it are adding more constraints to a problem that is already computationally challenging.

![Figure 7. Symmetric quadrilateral meshes of a collection of objects with several bilateral symmetries. The blue and red spheres represent singular vertices, which are themselves symmetric.](image)

**Code Complexity**

The majority of the quadrangulation methods are complex to code, and they must be maintained and supported by highly trained programmers. This is a major problem that is both slowing down the research in this area (because only a few research groups have access to the state-of-the-art quadrangulation algorithms) and drastically increasing the cost for companies to embrace these new technologies.

To ameliorate this problem, I recently released a full implementation of the global method I presented earlier, which is available in libigl, a simple C++ geometry processing library (https://github.com/libigl/libigl). The source code and binaries of the interactive retopology system SketchRetopo are also available online to foster future research in this area. An important venue for future work in this area is the design of alternative methods that are simple to implement and maintain, while having a quality comparable with the current approaches.

![Figure 8. Quadrilateral meshes created with the SketchRetopo system. The yellow curves have been manually designed by an artist, while the black quads are automatically created by the system. The yellow lines can be interactively adjusted, while seeing the effect on the final quadrilateral mesh in real time.](image)
Manual versus Automatic Quadrangulation

The ideal quadrangulation tool would allow a user to automatically tessellate a surface and then refine it by adding additional constraints that can be incorporated in the quadrangulation in real time. Unfortunately, this is not possible with any existing method. Completely automatic methods do not support exact constraints, and they are too slow to provide interactive feedback, while interactive methods cannot automatically quadrangulate an entire surface without user input. Combining the two approaches is far from trivial and would probably require a completely novel solution strategy, but it would provide the ideal solution for character retopology and CAD applications.

Volumetric Tessellation

The advantages of quadrilateral meshes over triangle meshes similarly extend to the volumetric case, where hexahedral meshes (such as sets of cubes) are preferred over tetrahedral meshes. This problem shares many similarities with quadrangulation, while being considerably more challenging. The singularities can be both vertices or segments inside the volume. This is an exciting topic, with many problems to solve and only a few problem-specific algorithmic solutions that cannot be applied to arbitrary shapes.

Quadrangulation as a Design Tool for Architectural Patterns

Many architectural applications require the design of semiregular patterns on surfaces, such as the design of flat panelizations, force-flow-aligned beam structures, and brick patterns. Only recently, researchers started to adapt existing quadrangulation methods to these problems, and I expect that a plethora of interesting problems and designs will be proposed as architects increasingly discover the potential of these novel design tools.

The advent of 3D printing and digital fabrication is increasing the demand for tools to quickly and efficiently acquire and edit geometric data. Although 3D modeling requires a considerable amount of skill and artistic education, the 3D acquisition and editing of models is a much simpler task accessible to the masses. The tools presented in this article are striving to streamline the geometry processing pipeline, with the goal of fully automating the conversion of a point cloud.
into a high-quality coarse control grid that can be used to edit and digitally fabricate variations of the scanned model. Many of the basic building blocks required to make this idea a reality are now available, and I look forward to the research challenges that lie ahead.

References


Daniele Panozzo is a senior researcher at ETH Zurich. Contact him at panozzo@inf.ethz.ch.

Contact department editor Jim Foley at foley@cc.gatech.edu.
The Future of Work

RICHARD MATEOSIAN

In this column, I look at a book that describes the author’s experience working for a company with essentially no physical offices and with workers all over the globe. He draws some conclusions about the future of work.


In the July/August 2010 Micro Review column, I briefly discussed Scott Berkun’s Confessions of a Public Speaker (O’Reilly, 2009), a book he wrote while trying to make a living as a talking head. But in the 1990s, Scott distinguished himself as a development manager at Microsoft, where he was instrumental in making Microsoft’s belated embrace of the Web and browsers successful. His other books qualify him to be called a management guru, so it was with trepidation that he stepped back into a management job.

The backstory

About the time my review of his previous book was published, Berkun was a WordPress blogger and a consultant to Matt Mullenweg, the creator of the WordPress blogging software and founder of Automattic (note the extra “t,” so the company name includes Mullenweg’s given name). Automattic runs wordpress.com, one of the most popular sites in the world. Approximately half of all WordPress-based blogs are hosted there for free. Mullenweg wanted to try a new organizational approach within Automattic. Partly as a result of Berkun’s advice, he split the company into 10 teams, and he invited Berkun to lead one of them. Berkun agreed to join the company as an employee. Going in, he made it clear that he would leave to write this book in approximately one year. He wound up staying for a year and a half, the last few months of which as a team member after recommending that one of his team members be promoted to succeed him.

The book tells a fascinating story—fascinating because of both the personal details and the company’s unique organization. In the early 1980s, I read Tracy Kidder’s Soul of a New Machine (Little, Brown and Co., 1981), and the personal side of Berkun’s book reminds me of Kidder’s story. Kidder was a reporter and not a participant, but he did see some of the same dynamics at work as the ones Berkun describes. The workers who were passionate about the goal made the project succeed by working behind the backs of the hard-driving project managers. At Automattic, there are no hard-driving managers, and everything is out in the open—almost painfully so—but passion and commitment are the prime motivators.

As a development project leader in the 1960s, I read John Kenneth Galbraith’s New Industrial State (Houghton Mifflin, 1967). Galbraith said many things in that book, but the one I remember nearly 50 years later is that in order to succeed, companies must abandon top-down decision making and recognize that management will increasingly lack the knowledge needed to make day-to-day operational decisions. In this era of agile organizations, that seems like a quaint insight, but getting from there to here was a long, bumpy ride. Automattic, as described in Berkun’s book, seems like the culmination of that journey.

A virtual company

In January 2003, Matt Mullenweg established the WordPress open source community by forking code from b2/cafelog, a GPL-licensed open source project whose founder had stopped supporting it. Mullenweg’s founding principles were transparency, meritocracy, and longevity. In August 2005, distressed about the existing options for deploying WordPress-based blogs, he founded Automattic with three community volunteers and no venture backing. They designed an antispam plug-in called Akismet—still one of the first things a new WordPress blogger installs—and used income from that to keep Automattic afloat until they could obtain more substantial financing. Toni Schneider joined the company as CEO in November 2005, and he and Mullenweg jointly managed a totally flat organization until they created teams in 2010, when the company had 60 employees.
Automattic has a simple business model. They sell upgrades to bloggers who want more than the many features they can get for free. They sell advertising on a few popular blogs, and they work special deals with premier clients like CNN, Time magazine, CBS, and NBC Sports, which host their websites on WordPress.com.

Because of the way the company started, it was completely natural for everyone to work where they pleased. While the company eventually acquired highly desirable premises on Pier 38 in San Francisco, employees rarely used them, though Mullenweg occasionally called on locally based employees to come in as props when media representatives or premier clients came calling.

Mullenweg regards remote working as ideal. It flattens everything, producing higher lows and lower highs—a generally more mellow experience. Automattic can afford to be a low-friction company because it supports the WordPress community and relies on satisfied customers. It feels little competitive pressure. It doesn’t need schedules because it doesn’t do marketing. It has minimal hierarchy, so decisions can be made with little fuss.

Most of the time employees communicated on Internet Relay Chat (IRC) and their team blogs (known as P2s). Although email was by no means prohibited, few Automattic employees used it, because it is closed. If you do not receive a copy of an email message, you have no way to find out about it. Every word ever typed on IRC or a P2 is archived and available to every employee.

The whole company held occasional all-hands get-togethers face to face in exotic places, and teams did the same somewhat more frequently. A tradition for these events, which usually lasted several days, was to decide on team projects to develop and publish before going home.

Berkun’s role

Berkun’s team was called Team Social. Its job was to invent things that made blogging and reading blogs easier. In his year leading that team, they developed Jetpack, a WordPress plug-in designed to make wordpress.com features available to WordPress-based blogs hosted on other sites. It’s the other first thing a new WordPress blogger installs. They also unified the commenting facilities of all WordPress blogs in order to integrate IntenseDebate, a popular commenting product that Automattic had acquired because it worked on other blogging systems as well.

The integration was called Project Highlander to suggest (a science fiction allusion) that it was a fight to the death between IntenseDebate and the other WordPress commenting facilities until only one survived. With 120 blog themes, WordPress had many different ways of making and presenting comments, and those had to be unified before Project Highlander could succeed.

Project Highlander called on project management skills that Berkun brought to Automattic from his days at Microsoft—skills that pushed Automattic in the direction of a more mature development process. This was a recurring theme of Berkun’s time there. In terms of Eric S. Raymond’s classic book The Cathedral and the Bazaar (O’Reilly Media, 1999), Automattic had grown up at the Bazaar end of the spectrum. Berkun, primarily because of his time at Microsoft, brought in aspects of the Cathedral approach whenever that was a more effective way to approach a problem. Automattic had 60 employees when Berkun joined and 170 by the time he finished writing this book, so some evolution in the Cathedral direction was inevitable, but Berkun’s expertise made it easier.

While embracing the Automattic way of working, Berkun also struggled against it. He had mastered the techniques of face-to-face interaction—maintaining eye contact, reading body language, detecting emotional nuance, and so forth. He had to learn to compensate for the fact that virtual interactions made most of those techniques nearly impossible to use. In fact, one of his accomplishments was to move team meetings out of IRC and into Skype video.

Another problem Berkun identified, but really had no answer for, is the dynamics of online threads. You might make a thoughtful post about an important issue and see no responses. You have no idea whether anyone has read it or is thinking about it. Or someone might react to one small point in your post, and the thread mutates to focus on that point rather than on the one you set out to direct attention to. Berkun raised this issue by posting about it, and the responses frustratingly exhibited the very problems he hoped to highlight.

Berkun liked the company culture of fixing things immediately, but he noted that people respond to the most recent problem, and if something doesn’t get fixed right away, it tends to be forgotten, regardless of its importance. Berkun tried to introduce a system of priorities that would make it more likely that tricky but important issues would not be swept under the rug. He hoped to engender more strategic thinking to go along with the company’s tactical mindset.

Berkun also tried to institute some sort of usability testing. The programmers who worked on WordPress features generally came from the WordPress community, so they had reason to feel that they understood their target audience, but Berkun was able to identify many areas where users had difficulties that simple design changes would alleviate.

High jinks

A major part of the story Berkun tells is about the people he worked with, how they worked together, and how they coalesced into an efficient team. Many of Berkun’s anecdotes concern his team’s meetings in places like New York, Seattle, and other, more exotic, places.

Seen from the outside, the team seemed like a bunch of hard-drinking young men, a few years out of college...
(more than a few in Berkun’s case), who enjoyed playing around the edges of trouble. For example, on the way to a bar in Athens after the one they’d been drinking in closed at 2:00 am, one team member miraculously escaped serious injury. On a dare, he jumped between 3-foot-high traffic bollards spaced 4 feet apart and missed his second jump, crashing toward the sidewalk. As Berkun describes it, “either through Australian training for drunk jumping or a special Krav Maga technique he’d learned, mid-fall he realized his predicament and managed to tuck and roll. …The silver-dollar sized patch of skin missing from his elbow seemed a fair price to pay, and he was glad.”

Despite this sort of incident, their meetings in exotic places were highly productive. Their time together seemed to fill a need that their usual distributed virtual interactions did not. Oddly, though, when working side by side, they often continued to communicate through IRC and their P2, as if they were continents apart.

Lessons

The first lesson learned from Automattic is that a virtual company can exist and be productive. It’s not the only such company; GitHub has a similar distributed structure. But Google, the dominant force in Silicon Valley, believes in colocation and with few exceptions requires employees to work in the office, not remotely. With Marissa Mayer’s move from Google to the helm of Yahoo, that meme has taken root at Yahoo as well. Many other Silicon Valley companies have also held that belief for years. Partly, they believe it’s a more efficient way to develop software, and partly they don’t trust their employees.

Trust is the key. Automattic believes in hiring great people, setting good priorities, granting authority, removing distractions, and staying out of the way. The way Automattic works makes it no harder to detect slackers than if you were looking over their shoulders every minute of the day. But most Automattic employees come from a tradition of working remotely on open source projects. They are self-sufficient and highly motivated, passionate about what they hope to achieve. Their way of working might not work for everybody, but it works for them. Berkun believes that Automattic has answered many questions that the working world is afraid to ask. Results trump traditions, and the most dangerous tradition is that work is both serious and meaningless, as exemplified by Dilbert. A short definition of work is “something I’d rather not be doing.” Automattic’s management—with its vision, mission, and long-term thinking—might be atypical, but it has given work meaning. Automattic’s workers have great freedom and take great pride in their work. And, as Berkun’s anecdotes show, they have a lot of fun.

This short and seemingly lightweight book actually contains a lot of meat, and I haven’t covered all of it here. If you’re interested in the future of work, you should read it.

Richard Mateosian is a freelance technical writer in the San Francisco Bay Area. Contact him at xrmxrm@gmail.com.

As wearable technology assumes an increasingly important function in daily life, sensors and other electronic devices applied directly to the skin, in forms like artificial nails and makeup, might further revolutionize human experience.

Beauty Technology: Body Surface Computing

Katia Vega and Hugo Fuks, Pontifical Catholic University of Rio de Janeiro

Up to now, my interest in artificial nails, false eyelashes, and makeup has been limited, to say the least. But when I first saw Katia and Hugo’s prototype for artificial nails with embedded RFID tags, I must say I was intrigued. Technically, the application is pretty straightforward, basically gluing RFID tags onto one’s fingers (much less exciting—but also far less controversial—than implanting chips into the body). Conceptually, however, the implications may prove quite significant.

Their Beauty Tech products’ easy implementation and convenience to affix on the body offer a particularly strong case for widespread adoption. Many people already wear artificial nails, so moving to digitally enhanced ones would be simple, probably even invisible. I believe the benefits derived from this kind of digitally enhanced product—for, example, being able to open a door or pay for coffee using, literally, your fingertips—are immediately obvious. With their widespread adoption, access and authentication technologies could become truly invisible in the very near future.

I hope this article inspires readers as much as I’ve been inspired and helps us all think more deeply about technologies coming into closer, more intimate contact with the human body. Another interesting question for me is whether Katia and Hugo’s beauty technologies might lead to more ubiquitous beauty product use (especially among the male population).

Albrecht Schmidt, column editor

As the revolution in wearable technology—miniature electronic devices attached to clothing and fashion accessories for both functional and purely aesthetic reasons—continues to flourish, why not extend the concept to include the human body itself? To this end, we propose beauty technology, a wearable computing paradigm that uses the body’s surface as a creative, interactive platform by integrating technology into beauty products applied directly to one’s skin, fingernails, and hair.

THE HUMAN BODY AS AN INTERACTIVE PLATFORM

Reliable miniaturized computing technologies have made possible a wide array of incredibly small wearable devices. Creating wearable products isn’t limited to tech companies like Google, Motorola, and Apple. Clothing industry icons like Nike and Adidas have also made significant investment in the wearable market, a clear suggestion that wearables are on the cutting edge in fashion design. The next logical step is to utilize the human body’s roughly two square meters of skin as a canvas for applying sensors and attaching other computing devices in ways that enhance the human experience.
Throughout the millennia, the skin—extending also into nails and hair—has played a crucial role as a protective barrier, a sensory monitor, a heat and moisture regulator, and an integral part of the body’s immune system. And since our earliest days on the savannah, we humans have applied various products to the skin to disguise, enhance, highlight, alter, and decorate our bodies. Yet despite becoming more sophisticated as advances in chemistry created new formulas and application processes, such beauty products have remained largely nonfunctional.

INVISIBLE COMPUTING, NOT CYBORG FASHION

Fully implementing Mark Weiser’s influential ubiquitous computing paradigm\(^1\) means creating technology that assists in our everyday lives by functioning invisibly and unobtrusively. Clearly, current functional wearable technologies fail to achieve this ideal because, like Google Glass, they are so obviously distinguishable from the human body. Moving closer to the body, however, and making the skin itself interactive, one can actually create invisible technologies. Our project proposes interfaces for cyborg empowerment but without the expected sci-fi appearance. Makeup and other similar beauty products might very well be visible, but they appear to function decoratively, as makeup always has, and not as a technology platform.

The project we envision modifies beauty products using widely accepted and broadly available technologies to create functional interfaces designed to attach on the body’s surfaces. We are already investigating embedded electronics for metalized eyelashes, special-effects (FX) makeup, and artificial nails.\(^2\)\(^3\) We also see opportunities for applications that can improve

**BEAUTY TECHNOLOGY FOR ACCESSIBILITY**

With Winkymote, we are exploring how beauty technologies can allow greater accessibility for people with disabilities. The application was inspired by Felipe, a 33-year-old master’s degree student who was injured practicing jujitsu and has been a quadriplegic for 13 years. As pictured in Figure A, Winkymote provides Felipe an infrared-controlled interface that uses FX e-makeup connected to an infrared-transmitting module mounted on a necklace worn around his neck. Based on Felipe’s voluntary winking, the switch closes, sending a digital signal from an infrared LED to the microcontroller that operates the functions of a television. It also activates an audio feedback informing Felipe that an appropriate signal has been sent. Blinking with his left eye, right eye, or both eyes, he can turn the TV on, turn it off, or change the channels up and down.

*Figure A. Winkymote, an infrared-controlled interface specifically designed to allow Felipe, a quadriplegic, to operate a television using voluntary eye blinks.*
accessibility for those with disabilities (see the sidebar “Beauty Technology for Accessibility”).

Conductive eye makeup
Change the world with just a blink. Wink your left eye, and turn on the lights; wink your right eye, and check your email. Beauty technology uses conductive eye makeup that senses voluntary blinking. Chemically metalized plastic eyelashes, as shown in Figure 1, are specially treated to maintain a natural color but at the same time act as switches. These send an electrical signal to a microcontroller via conductive materials attached to the skin as eyeliner. Then, specific tasks can be associated with winking either the left or the right eye or by closing both eyes at the same time—gestures that trigger activators on the wearable and other wirelessly connected devices.

Most people blink involuntarily 10 to 15 times per minute, and normal eye closure duration when blinking lasts from 150 to 250 milliseconds, with an upper limit in the 270 to 300 millisecond range. For purposes of triggering a signal, we defined voluntary blinking as eye closure lasting between 0.5 and 2 seconds. Based on this, we’ve created multiple applications that have been demonstrated in public presentations to confirm the concept’s practical feasibility.

For example, Superhero (http://superhero.katiavega.com) levitates objects with just a wink. The conductive makeup is connected to infrared emitters hidden in a headband. A wink triggers infrared commands, decoded from a remote control, in order to make a drone fly. Another application is Arcana (http://arcana.katiavega.com), a futuristic human “angel” who through her blinking alters the environment surrounding her. The conductive makeup is connected to a radio module hidden in the performer’s wig that communicates with a computer to activate music tracks and change the images being projected. This demonstration took place in an auditorium, with the projected images displayed directly on the actress.

From eyes to face: FX e-make up
Could your skin alone act as an interface? Can you control a computer, for example, using just your facial movements? Can we create beauty technologies that apply the activator directly to the skin? We explore these and other questions with Kinisi (http://kinisi.katiavega.com), an FX e-make up application that activates different light patterns through smiles, raised eyebrows, lip movements, and the like. Our FX e-make up represents a beauty technology prototype with sensors glued at precise points onto a latex facial mask. The mask, when applied to a human face, is capable of sensing muscle movements as the skin folds on the musculature of the face, acting in effect as a second skin. LEDs attached to the sensors demonstrate this sensing effect visually, as shown in Figure 2.

Beauty Tech nails
Radio frequency identification (RFID) and near field communication (NFC) microchip technologies integrated into plastic cards and keys—and increasingly emulated in cell phones—are widely used for access control and customer payment. But imagine opening a hotel room door or paying your bus fare without having to pull out a card, entering a movie theater without carrying a ticket, or shopping at the supermarket or checking out a library book without either a wallet or ID—just point and pay with literally everything at your fingertips.

As early as 1998, Kevin Warwick was experimenting with implanting RFID chips into human bodies (www.kevinwarwick.com), but most people find the idea of RFID implants scarily intrusive. However, attaching an RFID tag to one’s body in a semi-permanent way poses a much less threatening prospect. With Beauty Tech nails, we have created a fashionable, inexpensive wireless device that can attach to
fingernails with no external power components. These plastic or gel nails hide electronics such as RFID tags that create interfaces to identify the wearer.

Beauty Tech nails function like a conventional ID card into which an RFID tag is embedded. The wearer just passes her finger over a scanner, which then reads her unique ID, allowing contactless interaction with the interface. We have explored this technology’s utility for various entertainment applications. For example, Twinkle Nails (http://twinklenails.katiavega.com) interact with a musical box containing an RFID reader that translates each fingernail’s tag signal into a different note on the scale.

Other Beauty Tech nail devices can interact, almost magically, with water, sand, and wood interfaces: AquaDjing (http://aquadjing.katiavega.com), depicted in Figure 3, is a music controller that that allows a DJ wearing Beauty Tech nails to manipulate tracks by touching a water controller connected to a computer that creates sound effects and displays visualizations.

In addition to these more whimsical uses, Beauty Tech nails can replace conventional RFID cards for many, more practical applications.

We foresee wearable technology assuming an increasingly important role in daily life, making possible many novel interactions not previously imagined. As part of this trend, a variety of computerized applications, already available on clothing and accessories, can be applied directly to the skin’s surface as our project shows, transforming the body into an interactive platform with exciting new digital functionalities. Artificial eyelashes, makeup and cosmetics, acrylic nails—these all represent things people are used to seeing and having on their bodies. One can envision new prototypes that make use of these and other body-surface additions such as sensor tattoos and conductive hair extensions, all empowering wearers without giving them a cyborg look.

Our work so far is only a small step toward bringing computers into greater intimacy with the human body.

References
Katia Vega is a postdoctoral researcher in computer science in the Department of Informatics at Pontifical Catholic University of Rio de Janeiro, Brazil. Her research interests include beauty technology, software of places, wearables, and computer-supported collaborative learning. Contact her at katia@katiavega.com.

Hugo Fuks is an associate professor of computer science in the Department of Informatics at Pontifical Catholic University of Rio de Janeiro, Brazil. His research interests include beauty technology, software of places, wearables, and computer-supported collaborative learning. Contact him at hugo@inf.puc-rio.br.
Why do software engineering projects come in over-budget, yet under-optimized? What is it about the process that developers most often lose sight of—sometimes antagonizing their organizational constituents and bringing down management’s wrath?

One common answer is software engineering economics—that is, making decisions related to software engineering within a business context. Any project’s success depends partly on effective business management: systematically examining software attributes in ways that relate them to clear economic measures.

In essence, software engineering economics aligns technical development decisions with the organization’s larger business goals. Key aspects of software engineering economics include:

- **general finance concerns** to account for cash flow, valuation, inflation and depreciation, taxation, and the like;
- **life-cycle budgeting** based on understanding product-and-process life cycles, using portfolios and proposals, and considering time-value investment, pricing, costing, and strategies with regard to termination, replacement, and retirement; and
- **risk and uncertainty analysis** in terms of project prioritization, for-profit decision making, minimum acceptable rate of return on investment and capital, break-even evaluation, and cost benefit/cost effectiveness comparisons.

Mastering software engineering economics also involves understanding optimization, learning to apply the “good enough” principle, creating “friction-free” economic environments, maximizing use of software ecosystems, and knowing when offshoring and outsourcing are appropriate.

The IEEE Computer Society’s Professional Education division offers online review courses covering all these topics, and more, that can be applied to multiple real-world software projects. Open to members and nonmembers, these help prepare participants to gain the Software Economics Certification of Proficiency, a credential useful for professional growth as well as career advancement.

Interested in learning more about these courses, other courses leading to certification, or CS partner education providers? Visit www.computer.org/web/education/index.
CAREER OPPORTUNITIES

PLAID TECHNOLOGIES, INC. in San Francisco, CA seeks SrSwr Eng to dsgn, bd, test, & launch new swtr prdt w/ code Igly cmpsd in Javascript and Py- thon, Chrdg with expndng cmpx mcrtnt idntfication matching algirthms which invlv dvpng, creating, & modifyng the API swfrw & anlyzng user needs. Some invlvmt with visual dsgn & user intrctn. Resumes: Plaid Technologies, Inc, Attn: B. Khwaja 25 Maiden Lane, Suite 304, San Francisco, CA 94108.

ENGINEERING. Ruckus Wireless, Inc. has job opp. in Sunnyvale, CA: Cons-ultng Systems Engineer. Support Pre-Sales Account Teams on co. products. May telecommute from home office any-where in U.S. Mail resumes referencing Req. #CSE37 to: Attn: N. Enzminger, 350 W Java Dr, Sunnyvale, CA 94089.

SENIOR SOFTWARE DEVELOPMENT LEAD COMPANY/INSTITUTION: Limo-Link, Inc. Ad: LimoLink, Inc. has an opening for a Sr. Software Development Lead in our Cedar Rapids, IA office - Master's Degree or foreign degree equivalent in CS, CIS, Engineering or Math and experience with Visual Studio Team, ASP. Net, C#, SSRS, SSIS and SQL Server. (In lieu of a Master's Degree, employer will accept a Bachelor's Degree and 5 yrs. experience). To apply, emailcareers@ limolink.com or by regular mail : Hu-man Resources LimoLink Inc. 701 Tama Street, Building A Marion, Iowa 52302.

SAMSUNG SEMICONDUCTOR INC. has a Sr. Director, System Architecture (job code: 5FX0120) job opportunity in San Diego, CA: Analyze standards and spec-ififications, coming from 3GPP, operators and customers. Mail resume to Sam-sung Semiconductor, c/o Staffing – PTCL, 601 McCarthy Blvd., Milpitas, CA 95035. Must reference job code to be considered.

SOFTWARE ENGINEER - Design, Devel, Test & Implement large scale en-terprise & web apps using knowledge of OOP languages like C++, Core & Advanced Java, J2EE, EJB 3.0, Web-services (SOAP & REST),Spring Frame-work 3.0,Struts 2.0,Hibernate 3.0,JMS, HTML5, CSS3, Javascript, jQuery, Ajax, WebSphere ESB, Design patterns, WAS 7.0, Glass fish servers,DB2, SQL/PLSQL. Must be willing to travel & reloc. Reqs MS in sci, comp sci, eng or rel. Mail resumes to Strategic Resources International, 777 Washington Rd, Ste 2, Parlin, NJ 08859.

PROGRAMMER ANALYST (Mult. Open-ings) sought by Riviera Consulting, LLC in Parsippany,NJ w/a MS in S ware Enng or rtd. Candidate will provide apprc serv-ice mgmt across different envrnts by configuring an n-tier architecture (Web-server, Applic Server). This will help to deploy Java based apps on multiple domains. In addition, the Prgmr Analyst will perform tasks such as configuration, customization, performance tuning & load balancing, & maintenance of Web-logic Applic Service. Mail resumes to: Riviera Consulting, LLC, 239 New Rd, Ste B205, Parsippany, NJ 07054, Attn: HR.

TALLINN UNIVERSITY OF TECHNOLOGY

JOB DESCRIPTION
The Department of Informatics of Tallinn University of Technology, Estonia, EU, seeks to fill the position of a Full Professor in Information Society Tech-nologies (tenure-track position with dedicated funding from the Estonian Government).

The selected candidate will have an opportunity to join an effort by a multi-disci-plinary research team in building up the Competence Centre on Research in e-Government at Tallinn University of Technology. We expect from the person fulfiling the new position the integration and advancement of technologies utilized by information society and e-government and also teaching and cur-ricula development in that field.

Estonia is a worldwide leader in digital society and e-government. The coun-try is known as a pioneer in introducing digital services such as e-voting, e-signature, and e-taxation, and more recently, e-residency for foreigners. Tallinn University of Technology is a key enabler of e-Estonia. It is a public university established in 1918 that employs over 2000 staff supporting 14000 students. The Department of Informatics performs research in software en-gineering, information systems, e-government, data mining and fusion, and intelligent systems. The department is also running the International Master’s Program in E-Governance Technologies and Services. In building up the Competence Centre, our research team collaborates with the Estonian govern-mental ICT agencies.

REQUIREMENTS
We expect from the successful candidate:
• A PhD in information systems, computer science, informatics, or the equivalent, and an active interest in e-governance.
• A proven track record of the highest quality research in relevant areas.
• Excellent teaching skills and experience and a strong vision on academic education.
• Willingness to participate in curricula development.
• The vision and competences to initiate and manage research projects and supervise staff members and students.
• A proven track record in acquisition of research funds, a strong network
• A proven track record in acquisition of research funds, a strong network

Full Professor in Information Society Technologies (tenure-track)

Responsibilities:
• Successfully supervised doctoral students are an advantage.

Responsibilities:
• Applying information society technologies in areas such as information systems, cybersecurity, open data management, sociotechnical systems, public administration, and regulation.
• Teaching and designing courses in MSc curricula and particularly in the International MSc Curriculum in E-Governance Technologies and Services, and teaching courses at the PhD level.
• Preparing and running R&D projects funded by the EU and by the Estonian Research Agency and Enterprise Estonia.
• Contributing to consolidation and expansion of the department’s inter-national network via joint research projects with industry, governmental organisations, and universities.
• Supervising MSc and PhD students and postdoctoral researchers.

How to Apply:
Candidates are required to submit the following documents:
1) an application addressed to the Rector of the university;
2) copies of documents showing that the candidate possesses the required academic degree and education (in English);
3) a curriculum vitae and a list of publications;
4) vision/action plan of the research/teaching activities.

Documents should be sent to:
Personnel Department, Tallinn University of Technology
Ehitajate tee 5, 19086 Tallinn, Estonia
E-mail: personal@ttu.ee Phone: +372 620 2056
To receive full consideration, application and required materials should be received by August 15th, 2015
Contact person: Prof. Gert Jervan, Dean of the IT faculty (gert.jervan@ttu.ee)
SOLUTIONS ANALYST: Test, maintain, & monitor computer programs & systems. conduct unit testing & implementation of Microsoft SharePoint & .Net technology based applications. Expand/immodly system to serve business requirements. Utilize Agile (XP and Scrum), ASP.NET MVC, AJAX, CSS, JSON, SOA, XML, SOAP, & WCF. Will work in unanticipated locations. Req. 2 yrs exp. Send resume to Silicus Technologies, LLC 2700 Post Oak Blvd Ste 1625, Houston, TX 77056.


SIEMENS PLM SOFTWARE INC. has the following openings: Software Engineer Advanced Req#142380 in Milford, OH to create code to enhance & maintain the CMM; Infrastructure Engineer Advanced Req#142381 in Allen Park, MI to install, maintain & support Teamcenter software; Software Engineer Advanced Req#142401 in Cypress, CA to design, develop, modify & implement software programming for products. Email resumes to PLMCareers@ugs.com & refer Req#of interest. EOE.


SPLUNK INC. has the following job opportunities in San Francisco, CA:Senior Software Engineer, Systems & Infra-structure (Req#9D8U5H). Design, scale out, & maintain Co.s cloud-based infra-structure. Software Engineer (Req#8Z-8VLE). Design & develop apps supported by Co.s core platform. Software Developer (Req#9DE3UY). Design, implement, & deploy performance testing framework for Co. Cloud products. Software Engineer (Req#9AAUWN). Design, develop, & test core server sub-components in C, C++, Python, & shell scripting languages. Software Engineer in Test (Req#9SYVJA). SW design, analysis & development of SW testing tools to improve computer syst. UI/UX Designer (Req#9F7W48). Create user interface designs for Splunk products in the form of wire frames & detailed design specifications. Refer to Req# & mail resume to Splunk Inc., ATTN: J. Aldax, 250 Brannan Street, San Francisco CA 94107.

ENGINEERS: Axis, Inc., an established engineering services firm, is seeking Sr. Design Engineers; Design Engineers, and Design Engineers II. Sr. Design Engineer position

salesforce.com, inc. has the following positions open in San Francisco, CA:

Member of Technical Staff, Quality Engineering (Ref. #SP15S19): Perform functional manual and/or automated testing of features, including writing detailed testing plans and relevant test cases to cover business use cases, error handling and boundary conditions as defined in technical specifications.

Lead Member of Technical Staff, Software Engineering (Ref. #SP15S38): As a software development scrum team member, work with product management to gather requirements. Translate those requirements into an architecture and design for a software application, API, and/or frameworks.

Mail resume to salesforce.com, inc., P.O. Box 192244, San Francisco, CA 94119. Resume must include Ref. #, full name, phone #, email address & mailing address. salesforce is an Equal Employment Opportunity & Affirmative Action Employer.
SR. SOFTWARE ENGINEERS sought by Sagitec Solutions, LLC, an established global technology solutions company. Position requires Master’s degree or equiv. in Comp. Science, IT Engineering or related and 24 mos’ relevant work exp. in job offered or as a software designer, developer, tester, or analyst. Sagitec will also consider candidates with a Bachelor’s degree or equiv. in Comp. Science, IT Engineering or related and 24 months of work experience in job offered, or as a software designer, developer, tester or analyst. Position is based out of corporate headquarters in Little Canada, MN and subject to relocation to various unanticipated sites throughout the U.S. Mail resume to: Sagitec Solutions, LLC, Attn: Chief Human Resources Officer & Global Head-HR, 422 County Road D. East, Little Canada, MN 55117.

IGATE TECHNOLOGIES INC., an established IT consulting company with HQ in Bridgewater, NJ seeks qualified IT & Business professionals & managers (i.e., Operations Managers, HR Specialists, Financial Analysts, Sales Professionals, Computer & Network Systems Administrators, Computer Programmers, Database Administrators, Lead Software Developers, Software Architects (SAP), Software Engineers, Software Quality Assurance Engineers, Systems Analysts, User Interface/Experience Technicians & Web Architects) for its growing team. Operations Manager requires Bach or equiv in bus admin, IT, or related & 18 mos’ relevant indus exp. HR position requires Bach or equiv in Bus, HR, or related & 2 yrs’ relevant indus exp. Finance position requires Bach or equiv in Acctg, Finance, Bus, or related & 2 yrs’ relevant indus exp. Sales position requires Bach or equiv in bus admin, engineering, CS, or related & 2 yrs’ relevant indus exp. Lead Software Developer requires Bach or equiv in engineering, CS, or related & 4 yrs’ relevant indus exp. All other positions require a Bach or equiv in engineering, CS, or related & 2 yrs’ relevant indus exp. Positions are based out of Bridgewater, NJ HQ & may require relocation to various unanticipated locations throughout the US. Qualified applicants submit resumes noting desired position to HR Manager - Mobility, IGATE Technologies Inc., 99 Wood Ave South, Ste 800, Iselin, NJ 08830.

CISCO SYSTEMS, INC. is accepting resumes for the following positions:

AUSTIN, TX: SOFTWARE ENGINEER (REF.# AU52): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software.

COSTA MESA, CA: SOFTWARE ENGINEER (REF.# COS3): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of backend and client of digital TV software.

COLUMBIA, MD: SOFTWARE ENGINEER (REF.# COLU1): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software.

DENVER, CO: NETWORK CONSULTING ENGINEER (REF.# DEN3): Responsible for the support and delivery of Advanced Services to company’s major accounts. Telecommuting permitted and travel may be required to various unanticipated locations throughout the United States.

HOUSTON, TX: SYSTEMS ENGINEER (REF.# HOU6): Provide business-level guidance to the account team or operation on technology trends and competitive threats, both at a technical and business level. Telecommuting permitted.

RESEARCH TRIANGLE PARK, NC: SOFTWARE/QA ENGINEER (REF.# RTP4): Debug software products through the use of systematic tests to develop, apply, and maintain quality standards for company products. Software Engineer (REF.# RTP3): Responsible for the definition, design, development, test, debugging, release, enhancement or maintenance of networking software.

Customer Support Engineer (Ref.# RTP1): Responsible for providing technical support regarding the company’s proprietary systems and software. Virtual Consulting Systems Engineer (Ref.# RTP903): Engage customers virtually via collaborative tools and technology including web and video conferencing.

SAN FRANCISCO, CA: TECHNICAL SUPPORT ENGINEER (CNG STAFF) (REF.# SF7): Responsible for providing quality technical support for the company’s growing client and partner base. Travel may be required to various unanticipated locations throughout the United States.

SAN JOSE/MILPITAS/SANTA CLARA, CA: USER CENTERED DESIGN ENGINEER (REF.# SJ65): Responsible for the development of software artifacts to deliver software releases of the company’s services and products. Member of Technical Staff (Ref.# SJ150): Design and build capabilities, features and functions of cloud computing infrastructure offerings for hosted services and SaaS products across the company.

TEWKSBURY, MA: CONSULTING SYSTEMS ENGINEER (REF.# TEW11): Provide specific end-to-end solutions and architecture consulting, technical and sales support for major account opportunities at the theater, area, or operation level. Travel may be required to various unanticipated locations throughout the United States.

Please mail resumes with reference number to Cisco Systems, Inc., Attn: M51H, 170 W. Tasman Drive, Mail Stop: SJC 5/1/4, San Jose, CA 95134. No phone calls please. Must be legally authorized to work in the U.S. without sponsorship. EOE. www.cisco.com
Juniper Networks is recruiting for our Sunnyvale, CA office:

**QA Engineer Senior Staff** #3923: Deliver technically skilled testing methodology with Company’s Security Products for data center and Campus based networks. Provide technical guidance to testing engineers to achieve committed deliverables per each product release milestones.

**Hardware Engineer** #34413: Knowledge of schematic capture, logic design, printed circuit board design, and optical/electrical signaling. Experience with oscilloscopes and logic analyzers.

**Technical Support Engineer** #5260: Deliver high-quality technical assistance for switching products through telephone and electronic communication. Provide hardware and software technical support including configuration assistance and troubleshooting of mentioned products and other peripheral systems including networking equipment, servers, and clients for timely resolution of issues.

**ASIC Engineer Sr. Staff** #10899: Define and architect sub-system functionality, develop architectural specifications, and use the Verilog Hardware Description Language to design digital logic, based on architectural specification.

**Software Engineer** #33196: Design and develop software for network analytics on hypervisor, bare-metal-servers or network devices. Debug and fix issues in network analytics software as per the company software development process.

**Software Engineer** #30414: Develop detailed software functional and design specifications. Design, develop, unit-test and maintain embedded networking software.

Juniper Networks is recruiting for our Westford, MA office:

**Technical Support Engineer** #20407: Work with customers to resolve technical and non-technical problems related to routers, protocols and network design. Troubleshoot complicated hardware and software issues, replicate customer environments and network problems in the lab. Develop technical specialties and prepare technology white papers in these areas.

Juniper Networks is recruiting for our Herndon, VA office:

**Technical Support Engineer Staff** #11720: Provide high level expertise on company specific products. Deliver in-depth diagnostics and root-cause analysis for network impacting issues on routing products (Internet backbone routers) to large Internet Service Providers and/or enterprise customers.

Mail single-sided resume with job code # to:

Juniper Networks
Attn: MS 1.4.251
1133 Innovation Way
Sunnyvale, CA 94089

---

Intuit, Inc. has openings in Plano, Texas for:

**Staff Software Engineers in Quality** (Job code: I-244)

To use knowledge of software engineering best practices and principals to design, create, document, implement and/or maintain test scripts for complex on-demand and integration applications.

To apply, submit resume to Intuit Inc., Attn: Olivia Sawyer, J203-6, 2800 E. Commerce Center Place, Tucson, AZ 85706. You must include the job code on your resume/cover letter. Intuit supports workforce diversity.
COMPUTER PROGRAMMER: Write, update, & maintain computer programs & software packages. Create, modify, test code, forms, & script. Implement/support Oracle ERP(AP,AR,GL,PO,-INV,HRMS, iProcurement, SCM), P2 products. Utilize SQL,PL/SQL, OAF, ADF, Discoverer, BI Publisher, Hyperion, Java, J2EE, JDeveloper, SOA, SQL Server, SSIS, SSRS, Informatica, Unix, QlikView, Objective C, Cocoa Framework, Xcode. Familiar with oil & gas Production/Revenue concepts & infrastructure. Will work in unanticipated locations. Req. 2 yrs exp. Send resume to TecnicsConsulting, Inc. 2640 Fountain View Dr, Suite 200 Houston, TX 77057.

ENGINEER, FILEMAKER, INC. has an opening for the following in Santa Clara, CA: Product Marketing Engineer [Req#9PAMQ8] Pln & dsgn SW, iOS, and web user interfaces for FileMaker prdct line. Refer to Job#: 9PAMQ8 & mail resume: Apple Inc. ATTN: L.M. 1 Infinite Loop 104-1GM, Cupertino, CA 95014. Filemaker is an EOE/AA m/f/disability/vets.

SENIOR SOFTWARE DEVELOPMENT LEAD – LIMOLINK, INC. has an opening for a Senior Software Development Lead in its Cedar Rapids, IA office. Job duties: Planning and executing a variety of methodologies as part of the concept stage of the overall project development of three tier applications; creating GUI prototypes; researching, planning and developing project strategies; coordinating the full life cycle of IT projects using .Net framework, C#, ASP.Net, JavaScript, Visual Studio Team, VB Script, HTML, DHTML, XML, XSLT, LDAP, .NET Enterprise Services, WCF, Rest-based services, ADO.Net, TCP/IP and SMTP; coordinating the development and implementation of the applications; writing stored procedure and completing performance tuning using SQL Server, SSRS and SSIS; overseeing the implementation of test validations of the application; ensuring the optimization of the developed applications; participating in application walk throughs with users; producing project documentation; and resolving functional issues and support administrative activities for systems. Requirements - Master’s Degree or foreign degree equivalent in Computer Science, Computer Information Systems, Engineering or Math and one year’s experience in the job offered or one year’s experience in the IT field. (In lieu of a Master’s Degree, employer will accept a Bachelor’s Degree and five year’s experience). Special requirements - Experience with Visual Studio Team, ASP.Net, C#, SSRS, SSIS and SQL Server. Any Applicant who is interested in this position may email a resume to careers@limolink.com or apply by regular mail (including Reference Number 10001) to: Human Resources, Attn: Katy Thomas, Limolink Inc., 701 Tama Street, Building A Marion, Iowa 52302.

Help build the next generation of systems behind Facebook’s products.

Facebook, Inc. currently has the following openings in Menlo Park, CA (various levels/types):

Manager, Production Database (486EJ) Direct a team of engineers across different time zones to analyze and maintain Company’s service stability by documenting policies & best practices in daily, weekly, & annual-based operations.

Data Scientist (4062J) Apply your expertise in quantitative analysis, data mining, & the presentation of data to see beyond the numbers & understand how our users interact with our core products. Front End Engineer (FE315J) Work with Product Designers to implement the next generation of Facebook’s products. Build/design efficient & reusable front-end abstractions/interfaces/systems.

Research Scientist (3325J) Research, design and develop new distributed system software architectures, data structures, algorithms & techniques to improve the efficiency & performance of Facebook’s platforms.

Facebook, Inc. currently has the following openings in New York, NY (various levels/types):

Research Scientist (3502J) Research, design, & develop new optimization algorithms & techniques to improve the efficiency & performance of Facebook’s platforms.

Mail resume to: Facebook, Inc. Attn: JAA-GTI, 1 Hacker Way, Menlo Park, CA 94025. Must reference job title and job# shown above, when applying.
Apple Inc. has the following job opportunities in Cupertino, CA:

Web Developer (Req#9AHVF4). Manage & enhance customer-facing Java apps. Administer servers which host reporting dashboards. Troubleshoot system/application issues.

Software Engineer Applications (Req#9GWRDR). Configure & maintain systems for cloud computing services. Maintain & upgrade systems in production on Solaris, Linux, & other specialized appliances.

Hardware Development Engineer (Req#9F4QW9). Lead development & qualification activities to design & develop innovative Flat Panel Displays used in Mac products.

Software Engineer Applications (Req#9EGS6Y). Resp for stdrdizing & simplifying SW sys. Provide prod support for iCloud services & back-end apps.

RFIC Design Engineer (Req #9BV3LG). Respn for the design of radiofrequency integrated circuits for wireless comms.

Software Development Engineer (Req#9CYTCK) Res, des, dev, imple & debug statistic & determine natural lang process SW as part of nov ttxt input sys.

Software Engineer (Req#9MKTB4) Design data models for storage large amnt of data.

ASIC Design Engineer (Req#9AJUWL). Perform phys design & implmnt partitions. Perform placement, clock tree synthesis & routing on the design.

ASIC Design Engineer (Req#9QNZT). Design test logics & genrate test patterns for app processors.


Hardware System Integration Engineer (Req#9DZUWX). Des & dev HW for Mobile Devices. Run simulations for high speed design rules, power layout & integrate with the final des. Travel req’d 15%.

Software Development Manager (Req#9CPNXF). Des, manage and track build SW architect that meets cust require.

Software Quality Assurance Engineering Manager (Req#9LKTN4). Manage team of 3 SW QA Engs for the iAd Serving Syst.

ASIC Design Engineer (Req#9D8VSF). Implement physical design of partitions for highly complex SOC utilizing state of the art process technology.

ASIC Design Engineer Applications (Req #9LL3FB). Respn for design, develop & deploy of data warehouse solutions for multi bus groups.


Software Engineer Applications (Req#9E52XE). Respn for SW apps & frameworks used across Apple internal & customer-facing apps.

Test Design Lead (System Design Engineer) (Req#9J3NR9). Analyze & coordinate the sys design functions of the TDL Engineering Group. Travel req’d 20%.

Software Development Engineer (Req#9F4SQL). Respn for crafting the user interface design of iOS & OSX.

Software Engineer Systems (Req#9MHVDY). Dvlp & maintain an online reporting app.

Software Development Engineer (Req#9E62RY). Audit & review SW architecture & source code to identify security vulnerabilities & potential enhancements.

ASIC Design Engineer (Req#98NVGW) Validate & characterize analog IP used in silicon chips for mobile apps.

Sr. Data Analyst (Req#9M93ET) Des, dvlp & analyze Data Services & Data Analysis Tools.

Engineering Project Lead (Business Data Architect) (Req#9AYUM4). Design & allocate people & lab space environments.

Systems Design Engineer (Req#9AJN4L) Perform parametric eval & front end design of radio frequency (RF) products such as cell phones, laptops & tablets.

Software Engineer Applications (Req#9N4UZL). Dev SW on iOS & web platforms for client (user facing) apps related to fraud analytics, search, HW diagnostics, & social networking.

Software Engineer Systems (Req#9H7TN3). Design, develop, debug & test SW for camera subsystems in Apple’s products.

Software Development Engineer (Multiple Positions Open) (Req#9H4MAW). Design & dev SW & firmware for an 802.11 WiFi stack running on a mobile platform.

Software Engineer Applications (Req #9MY4LM). Respn for presenting unified set of worldwide POI avail in Apple Maps using data sci & machine learn to conflate data from multi sources.

Mechanical Design Engineer (Req#9BRRU8). Des & dev textiles & fabrics. Dev & impl quality materials plans for suppliers & vendors. Travel req’d 35%.

Systems Design Engineer (Req#9KA38K) Des & dvlp automation tools for cell testing of iOS devices.

ASIC Design Engineer (Req#9B3NC7) Res for physical design & implementation of partitions.
Software Quality Assurance Engineer (Req#9HDTUU). Write, maintain, & run automation test scripts. Integrate testing scripts with automation framework.

Software Development Engineer (Req#9HMW5). Res for the des & dvlpmnt of advanced map imaging SW.

Software Quality Assurance Engineer (Req#9LUSD9). Create, dev, & exec all test strategies to ensure highest quality embedded prod SW sol.


Hardware Development Engineer (Req#9E623Y). Dsgn & dev RF HW for future Apple product w/ cellular & wireless capability.

Software Development Engineer (Req#9CXTY2). Respon for design & implement new feats of iOS productivy, apps (Contacts & Notes).

Software Engineer, Applications (Req#9GJ5K7). Dsgn & dvlp SW for the iTunes App Store.

Software Engineer Applications (Req#9SY2NW). Design & develop automation frameworks for existing & new apps.

Systems Administrator (Req#9GHU78). Provide technical support for storage, network, & compute infra-structure to the Hardware Technology team.

Software Engineer Applications (Req#9PP92EU). Architect solutions while playing a design & dvlpmnt role to deliver products in a highly available, scalable & integrated environment.

Software Developers, Systems Software (Req#9CZTVH). Devlpmnt of SW & HW for survey camera sys-tems. In depth data anlysis, incldng image analysis and flight data anlysis.

Software Engineer Systems (Req#9H4P5N). Respon for design, develop, implement, & debug of input device sw.

iOS Performance Engineer (Req#9QAQ5U). Analyze & improve iOS sw using performance analysis tools.

Hardware Development Engineer (Req#9SU48L). Spprt dsgn of new bat technologies thru use of electro-chem models.

Software Development Engineer (Req#9M4VT2). Dev digital video & audio SW.

ASIC Design Engineer (Req#9L5NF3). Conduct transistor-level feasibility studies for various systems/cchts.

Software Engineer Applications (Req#9H4NPQ). Design, dev & deploy data warehouse solutions for multiple business groups at Apple.

Software Development Engineer (Req#9FNUW). Quality latest sync techs for iOS & iCloud on Windows.

Software Engineering Applications Manager (Req#9E5VLU). Direct activities of SW apps dvlpmnt in the delivery of business intelligence solutions.

Software Development Engineer (Req#9CZQK8). Des & dev media SW for embed sys.

Senior Software Engineer (Req#9L5M5N). Design & dv lp major features & work jointly with team members to deliver complex changes.

Software Engineer Applications (Req#9NP3NE). Design & dev scalable, high qual machine learning platforms to service a variety of internal & external products.

Software Engineer Applications (Req#9FDVT3). Design & dev SW for app deployment automation.

Systems Design Engineer (Req#9DCT76). Eval the latest iPad, iPhone & iPod HW sys. Perform early evals of prototypes & HW sys on different wireless technlgies. Travel req’d 30%.

Software Quality Assurance Engineer (Req#9T7U27). Perform bus req analysis & create detailed test plans for large scale & high volume financial retail system.

Software Development Engineer (Req#9F2ZND). Design & develop computer vision algorithms for cam-dera features.

Refer to Req# & mail resume to

Apple Inc., ATTN: L.M.,
1 Infinite Loop 104-1GM,
Cupertino, CA 95014.

Apple is an EOE/AA m/f/ disability/vets.

Apple Inc. has the following job opportunity in Grapevine, TX:

Systems Design Engineer (Req#9C32E4). Test, debug, & execute wireless cellular products.

Apple Inc. has the following job opportunity in Culver City, CA:

Software Development Engineer (Req#9LUTMB). Dsgn & dvlp digital signal processing (DSP) software & algorithms for audio & voice sys.
Intuit, Inc.
has openings in
Mountain View, CA for:

Software Engineers 2
(Job code: I-47)

To design and implement highly scalable and performant components of a service oriented architecture platform in the payments domain, and work on SOA based platform that enables merchants and developers to accept payments.

To apply, submit resume to Intuit Inc., Attn: Olivia Sawyer, J203-6, 2800 E. Commerce Center Place, Tucson, AZ 85706. You must include the job code on your resume/cover letter. Intuit supports workforce diversity.

Samsung Semiconductor Inc.
has the following job opportunities in San Jose, CA:

Sr. SW Development Engineer
(Job code: SBR1917)

Conduct key research and development activities for SSD/flash memory technologies or other future memory.

Staff Engineer
(Job code: SAV1919)

Design analog integrated circuits such as Phase Locked Loop, Phase Interpolator, Clock Distribution circuits, Analog front-end, etc.

Staff Research Scientist
(Job code: SAL0206)

Lead and perform the research on the advanced transport physics in the sub-20nm FinFET and nano-wire devices.

Mail resume to Samsung Semiconductor, c/o Staffing – PTCL, 601 McCarthy Blvd., Milpitas, CA 95035. Must reference job code to be considered.
Call for Presentations

12 October - 15 October
Hilton Long Beach, Long Beach, California, USA

STC is a practitioner’s conference focused on Meeting Real World Challenges through Software and Systems Technology. As technologists and as citizens, we are faced with myriad challenges from defending national security, to ensuring the robustness of our critical infrastructure, to sustaining and enhancing large portfolios of legacy systems - all within ever tighter resources constraints.

We invite you to connect with your colleagues and share your ideas, success stories, and lessons learned in applying software and systems technology to meet our common challenges, by submitting an abstract for consideration as one of our conference session or tutorial speakers.


Focus areas this year include:

- Critical Infrastructure Challenges
- Agile/Lean Development
- Cybersecurity
- Supply Chain Risk Management
- Cloud computing
- Mobile applications
- Big Data and Data Analytics
- Affordability
- Open source
- Systems engineering challenges for software-intensive systems
- Human capital and workforce knowledge transition

Important Dates (**Updated**)
June 15, 2015 - Abstract submission closes
July 31, 2015 - Acceptance notices sent
August 14, 2015 - Final abstract and Bio submission
September 11, 2015 - Final presentation due
Focus on Your Job Search

IEEE Computer Society Jobs helps you easily find a new job in IT, software development, computer engineering, research, programming, architecture, cloud computing, consulting, databases, and many other computer-related areas.

New feature: Find jobs recommending or requiring the IEEE CS CSDA or CSDP certifications!

Visit www.computer.org/jobs to search technical job openings, plus internships, from employers worldwide.

http://www.computer.org/jobs
extravaganzas and progressed from tallying high scores to tracking persistent profiles. Interfaces have morphed from devices with one joystick and a single button to haptic feedback motion controllers and touchscreens. We can tell some new systems to turn on, and they will (although if a machine will turn on when you tell it to, doesn’t that mean that it was already on?). Our gaming libraries have expanded from a few cartridges next to the TV to vast compendia of digital content (a Steam library with hundreds of games is not uncommon). And coming next? Virtual reality.

Witnessing this gaming evolution has given me a slightly biased—dare I say “grumpy”—perspective on the world of games my son is only beginning to venture into. Back in my day, we couldn’t just buy a new truck to race around the track; we had to earn it through skill (and, sometimes, sacks and sacks of quarters). Back in my day, we didn’t have sprawling CGI cutscenes; we had a little 8-bit sprite running around, with two channels of 1-bit monaural sound providing ambience. Back in my day, we had to find a directory in DOS and type the .exe file to launch a game; now, we just click on the app.

Foreign Languages

I remember trying to teach my stepdad a baseball game on my old Nintendo system. He had played baseball for the University of Georgia; he was All-SEC one year and even went on to the minor leagues briefly (back when working on the family farm was more lucrative).

He sat down in my room, picked up the controller … and was instantly baffled by the directional pad and two buttons. I tried to explain how to swing the bat (I think it was not much more than “press the A button”), but it proved too complicated. Together in our backyard, he could show me the grip required to throw a knuckleball, the subtleties of swinging a bat depending on whether I was aiming for power or just trying to make contact. But the simulated version of the sport? It was like learning a different language. He gave up.

That language comes naturally to kids these days. Touchscreens are the gateway drugs of the videogame world. Before his second birthday, my son had demonstrated to my mom how to use an iPad (she’s got it down now, though; Match-3s and word search apps fill her phone and tablet—but that’s another column). Mason would occasionally play around with a monster truck game on our iPad, and the first time he showed me how to replicate a bug, I was wonderstruck. (For reference, the truck in question had to drive to the end of an arena and up to a wall, then back up, then drive to the wall again, and finally turn its wheels without pushing the pedal. Then it would fall through the floor and then the ceiling. Craziness.)

I started him up with some old Nintendo games, played through an emulator; he liked that enough (laughing every time Mario jumped and hit the coin blocks with his head). But when we moved to racing games, he started getting serious. And that’s where we’ve been for the past few weeks. We’ve had a couple other games enter into the mix, and at one point recently Mommy and Daddy were both playing with him. It was at that moment I realized I didn’t open the door to videogames for Mason just so he can play them; I did it so we can speak the same language.

That language is sometimes filled with violence (I could fill up too many pages with my problems about the sheer amount of violence in some games) and sometimes filled with predatory pricing models and behavioral psychology–influenced design choices. Online gamers can be rude, racist, and sexist. But for now, just as they were for my brother and me all those years ago, games are just another way to spend time together.

And at some point in the nearish future, maybe Mason and I will venture into a VR-based racing simulator together. Then we’ll see how long it’ll take before he can beat his old man.

Brian Kirk is IEEE Computer Society’s associate manager for editorial project development. Contact him at bkirk@computer.org.
Boost! Boost! Watch out for the rocks! Oof. Nice one! Oh … sorry, but we’ve recently added a new activity to our daily family routine: video games.

As the parent of a five-year-old, I didn’t come to this decision lightly. Still, I think there are plenty of reasons why it’s a good idea.

I, Gamer

We had a Pong system when I was a kid; I still remember the magic of that little box with switches. My brother and I played it a lot, watching the sticks on the screen move up and down. A few years later, we upgraded to an Atari 2600, and we would get lost for hours at a time, jumping on alligators’ heads, shooting AT-ATs, and trying to figure out exactly what was going on with E.T.

I enjoyed video games throughout my childhood, and I used my first paycheck ever to buy a Super Nintendo system. I bought a PlayStation when I graduated high school, and when I moved from one coast to the other, my PlayStation 2 was one of the few possessions I packed in my car. I can track major life moments in my past by what games I was into, much as other people contextualize through music, movies, and television.

And I’m not alone. Reports estimate the number of gamers in the world at over 1 billion, with around 700 million people playing online games in some form at any given time. No longer relegated to a console in the house, games now follow us around, stored in these magic little phones we carry. How could I not start gaming with my kid?

Pixels, Pixels Everywhere

I’ve been lucky enough to witness some monumental changes in video games and gaming. Beyond Pong and our Atari, my brother and I marveled at the aural and visual sophistication of games in the arcades. If I were to look at these today, of course, they’d look like amateur, blocky approximations of something fun. Back then, we were dealing with RAM measured in bytes, a maximum number of colors in the hundreds, and RCA connectors for output. Today, depending on your gaming rig of choice, you could see dedicated video RAM easily in the 2-Gbyte range, with millions of colors, HDMI output, and processing power strong enough for scientific work.

We’ve gone from clunky-looking, two-dimensional heroes to cinematic, big-budget

Cont. on p. 71
Instant Access to IEEE Publications

Enhance your IEEE print subscription with online access to the IEEE Xplore® digital library.

- Download papers the day they are published
- Discover related content in IEEE Xplore
- Significant savings over print with an online institutional subscription

Start today to maximize your research potential.

Contact: onlinesupport@ieee.org
www.ieee.org/digitalsubscriptions

“IEEE is the umbrella that allows us all to stay current with technology trends.”

Dr. Mathukumalli Vidyasagar
Head, Bioengineering Dept.
University of Texas, Dallas
Win the New Cybersecurity War with the New Rock Stars of Cybersecurity

Cybercrime is no longer a matter of credit card breaches. Cybercriminals are now trying to take down countries as well as top companies. Keep your organization safe. Come to the premier, one-day, high-level event designed to give real, actionable solutions to these cybersecurity threats.

Learn from and collaborate with the experts—

27 October 2015
The Fourth Street Summit Center
San Jose, CA

REGISTER NOW
Early Discount Pricing Now Available!

computer.org/cyber2015

CHRIS CALVERT
Global Director, HP Enterprise Solutions Products

MARCUS H. SACHS
VP, National Security Policy, Verizon

DR. SPENCER SOOHOO
CSO/Director, Scientific Computing, Cedars-Sinai Medical Center